Intersection Types, λ-models, and Böhm Trees

Mariangiola Dezani-Ciancaglini, Elio Giovannetti, Ugo de'Liguoro

Dipartimento di Informatica, Università di Torino,
Corso Svizzera 185, 10149 Torino (Italy)

Abstract

This paper is an introduction to intersection type disciplines, with the aim of illustrating their theoretical relevance in the foundations of λ-calculus. We start by describing the well-known results showing the deep connection between intersection type systems and normalization properties, i.e., their power of naturally characterizing solvable, normalizing, and strongly normalizing pure λ-terms. We then explain the importance of intersection types for the semantics of λ-calculus, through the construction of filter models and the representation of algebraic lattices. We end with an original result that shows how intersection types also allow to naturally characterize tree representations of unfoldings of λ-terms (Böhm trees).

1 Introduction

Pure λ-calculus [15, 54, 90], as is well-known, formalizes the algorithmic notion of function, which – in contrast to the set-theoretic notion – does not need explicit definitions of domain and codomain. However, computational objects come naturally with some kind of type distinctions, and concrete algorithms are generally intended to operate on certain types (i.e., set-theoretically, domains) of inputs to produce certain types of outputs; generic algorithms – say, a sorting algorithm – may operate on different concrete types of input, giving results whose type may depend on the input types (e.g., the sorted sequence will be of the same type of the original sequence).

These aspects of algorithms are described by λ-calculi endowed with type systems, where, in addition to λ-terms, one also has type expressions, built from atomic types and type constructors. λ-calculus being a theory of (higher-order) functions, the essential constructor is the arrow. For what concerns atomic types, they can be limited to (an infinite set of) type variables, since many properties of algorithms in relation with types may be studied without introducing predefined basic types with their constant names int, bool, etc., somehow as in pure λ-calculus the general theory of functions is better done without individual (integer, boolean, etc.) constants with special reduction rules (i.e., δ-rules).

There are basically two kinds of type systems [16]: the systems à la Church [24], also called typed calculi, and the systems à la Curry [37, 38], also called type assignment systems. In typed λ-calculi every term comes with its own type, and the same holds for all its subterms, starting from variables; for each type (i.e., type expression[1]) σ the existence of an infinity of typed variables x^σ, y^σ, z^σ, ... is assumed. Legal terms are only the well-typed terms, i.e. those built following the typing rules; types are integral parts of terms, though subterm types are often omitted in practice. For example, the simplest of such systems [90] consists of two rules only, for the application and the abstraction respectively:

1. if $M^{\sigma\to\tau}$ and N^σ are (well-typed!) terms, then $(M^{\sigma\to\tau}N^\sigma)^\tau$ is a term;
2. if M^τ is a term, then $(\lambda x^\sigma.M^\tau)^{\sigma\to\tau}$ is a term.

As is generally observed, the above type system is too restrictive, in the same sense as the type discipline of the programming language Pascal proved to be too rigid. It forces algorithms that may operate uniformly on a variety of types to split into an infinity of typed functions, like the standard example of the identity $\lambda x.x$, which splits into an infinity of typed identities $(\lambda x^\sigma.x^\sigma)^{\sigma\to\sigma}$, one for each possible type σ.

A type assignment system, on the other hand, is a system of typing rules by which terms of the pure untyped λ-calculus may be assigned (one, or more generally, infinite) types. Thus an untyped term, instead of splitting into different typed terms, merely turns out to be typable with different types; for example, the identity will be typable with all types of the form $\sigma\to\sigma$, such as $\varphi\to\varphi$, $(\varphi\to\chi)\to(\varphi\to\chi)$, etc. In general, not all terms will be typable, and different systems may differ both in the class of terms that are typable and in the types that can be assigned to typable terms.

The assignment of a type to a term is a statement of the form $M:\sigma$, which may depend on assumptions about the types of free variables. It may therefore be written as a judgment of the form $\Gamma\vdash M:\tau$ where Γ, which is usually called a type basis or a type environment, is a finite set of assumptions of the form $x_1:\sigma_1, x_2:\sigma_2, \ldots, x_n:\sigma_n$, with $x_1,\ldots,x_n$ all distinct. The notation is justified by the analogy with the judgment that a proposition σ is derivable from the assumptions $\sigma_1,\ldots,\sigma_n$. A type assignment system may then be viewed as a natural-deduction inference system [80], consisting of at least one axiom for the variables, and two rules for application and abstraction respectively:

$$\Gamma, x:\sigma\vdash x:\sigma\ (\text{var}) \qquad \frac{\Gamma\vdash M:\sigma\to\tau \quad \Gamma\vdash N:\sigma}{\Gamma\vdash M:\tau}\ (\to\mathrm{E}) \qquad \frac{\Gamma, x:\sigma\vdash M:\tau}{\Gamma\vdash\lambda x.M:\sigma\to\tau}\ (\to\mathrm{I})$$

The above axioms and rules are identical to the inference rules of the implicative propositional logic, and illustrate in its simplest form the well-known Curry-Howard correspondence between types and propositions [37, 38, 58, 91]. Types correspond

[1]We will denote type variables by the Greek letters φ, χ, ψ, and general type expressions by the Greek letters σ, τ, ρ, possibly with subscripts.

to propositions, terms are codings of natural-deduction derivations, β-reduction is cut elimination (in the extended sense of proof normalization step both for sequent calculi and for natural deduction, where sometimes it is called, more properly, detour elimination [80]).

In a type assignment system limited to the above axiom and rules all the typable terms are (strongly) normalizable [89]; equivalently, all the proofs in the above logic are reducible (using an arbitrary reduction strategy) to normal forms (i.e., proofs without cuts or equivalently without detours).
In other words, every computer program represented by a typable term is guaranteed to terminate, with any computation strategy. However, the class of λ-terms (and therefore of programs) typable in this system is very restricted, since it corresponds to the generalized polynomial functions of natural numbers.

If this type assignment system is compared with the simply typed λ-calculus à la Church, it can easily be seen that they actually are two notational variants of the same system, in the precise sense that a term M is typable à la Curry with a type σ if and only if there exists a term M' à la Church that is typed with the same type σ and is identical (when types are erased) to M [16].
The distinction between the two kinds of systems would thus appear inessential. However, this is in general no longer true for more complex systems, where there may not be such a natural correspondence between typed calculi and type assignment systems, so that the distinction makes perfectly sense.

The propositions-as-types analogy can be trivially extended from the implication to the logical conjunction [60] (which we denote by the symbol &), if we enrich the pure λ-calculus with the pair constructor and selectors, along with their special reduction rules:

$$\mathsf{fst}\langle M, N\rangle \to M \qquad \mathsf{snd}\langle M, N\rangle \to N.$$

The rules of &-introduction and &-elimination exactly correspond to the typing rules for the new constructs, and the logical conjunction $\sigma \mathbin{\&} \tau$ of propositions exactly corresponds to the cartesian product $\sigma \times \tau$ of types:

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N\rangle : \sigma \mathbin{\&} \tau}\,(\&I) \qquad \frac{\Gamma \vdash M : \sigma \mathbin{\&} \tau}{\Gamma \vdash \mathsf{fst}\, M : \sigma}\,(\&E) \qquad \frac{\Gamma \vdash M : \sigma \mathbin{\&} \tau}{\Gamma \vdash \mathsf{snd}\, M : \tau}\,(\&E)$$

There is, however, another sense of the word "and", directly in the language of types and unrelated with the ordinary Curry-Howard correspondence: we may want to express within the system that a term may be assigned both a type σ and a type τ [28]. The introduction and elimination rules that naturally arise are therefore:

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash M : \sigma \wedge \tau}\,(\wedge\mathrm{I}) \qquad \frac{\Gamma \vdash M : \sigma \wedge \tau}{\Gamma \vdash M : \sigma}\,(\wedge\mathrm{E}) \qquad \frac{\Gamma \vdash M : \sigma \wedge \tau}{\Gamma \vdash M : \tau}\,(\wedge\mathrm{E})$$

The new type constructor does not correspond [52] to any logical connective in the ordinary propositions-as-types analogy, which refers to minimal or intuitionistic logics, and must therefore not be confused with the above mentioned conjunction-type as cartesian product type. As a matter of fact, in proof-theoretic terms the meaning of the formula $\sigma \wedge \tau$ is not merely that there are proofs for the two propositions σ and τ, but that the two proofs are identical (since the term M, which codes the proof of σ, also codes the proof of τ) [65]. At the same time, since the introduction rule does introduce a new type construction, i.e. a new composed proposition, but does not introduce a new term construction, the correspondence between terms and proofs no longer holds in a strict sense. A propositions-as-types correspondence has recently been found with a form of relevant logic [93], but we will not discuss it here.

The type assignment system obtained by adding to the rules for the arrow the rules for the intersection turns out to possess the remarkable property of exactly characterizing the class of the strongly normalizable λ-terms, i.e., a term is strongly normalizable if and only if it is typable in this system [77].
Moreover, since the notion of intersection type is based on an intuitive set-theoretic interpretation, the introduction of a universal type ω, representing the set of all terms, is rather natural [33]. This meaning of the new and only type constant can be expressed formally by a new typing rule – or better, axiom – which states that every term may be assigned the type ω:

$$\Gamma \vdash M{:}\omega \qquad (\omega)$$

It turns out that the type system resulting from the system with intersection types by the addition of the rule (ω), where by definition all terms are typable, allows a straightforward type characterization of the class of normalizable terms, i.e. the terms possessing normal forms, i.e., the terms representing the total recursive functions.
Namely, normalizable terms are exactly those that can be typed with a type having no occurrences of ω, though of course ω may occur in the derivation itself [33] (if it does not occur in the derivation either, the derivation is actually in the restricted typing system without ω, and the term is therefore not only normalizable but in particular strongly normalizable [77]).
Finally, the system with ω also allows, as we will see, a simple type characterization of terms possessing head normal forms [33].

The complete type assignment system, which is known in the literature as the system CDV, is reported, for the reader ease, in Table 1, along with the syntax of type expressions. The system without rule (ω) will be referred to as the system $\mathrm{CDV}_{\not\omega}$.

1.1 Summary

Sections 2 to 9 are devoted to the proof-theoretical properties of intersection type assignment systems and to their power in characterizing normalizing and strongly normalizing pure λ-terms. They are kept at an informal level of exposition since our aim is to introduce the reader to the main ideas behind these systems. Therefore we avoid details of complex syntactical proofs, giving hints toward them and referring the reader to the literature.

The set $\mathcal{T}$ of types is defined by:

$$\sigma := \omega \mid \varphi \mid (\sigma \to \sigma) \mid (\sigma \wedge \sigma)$$

where φ ranges over a denumerable set $\mathcal{TV}$ of type variables.

The Type Assignment System

$$\Gamma, x{:}\,\sigma \vdash x{:}\,\sigma \ (\text{var}) \qquad \frac{\Gamma \vdash M{:}\,\sigma \to \tau \quad \Gamma \vdash N{:}\,\sigma}{\Gamma \vdash MN{:}\,\tau}\,(\to\text{E}) \qquad \frac{\Gamma, x{:}\,\sigma \vdash M{:}\,\tau}{\Gamma \vdash \lambda x.M{:}\,\sigma \to \tau}\,(\to\text{I})$$

$$\frac{\Gamma \vdash M{:}\,\sigma \quad \Gamma \vdash M{:}\,\tau}{\Gamma \vdash M{:}\,\sigma \wedge \tau}\,(\wedge\text{I}) \qquad \frac{\Gamma \vdash M{:}\,\sigma \wedge \tau}{\Gamma \vdash M{:}\,\sigma}\,(\wedge\text{E}) \qquad \frac{\Gamma \vdash M{:}\,\sigma \wedge \tau}{\Gamma \vdash M{:}\,\tau}\,(\wedge\text{E})$$

$$\Gamma \vdash M{:}\,\omega \qquad (\omega)$$

Table 1: The system CDV

Sections 10 to 12 treat the semantics of type assignment. Here we give a heuristic exposition, centered on the justification of the filter model construction. More technical parts are also given, but often only sketched.

Finally, Section 13 concerns the tree representation of the unfolding of a λ-term and its characterization via typings. This is the original contribution of the present paper.

We conclude with suggestions of further readings in Section 14.

2 Some Examples of Typings

If we identify each type with the set of terms that – w.r.t. a sufficiently large (actually infinite) basis – are typable with that type, and we thus read the colon as a membership symbol, the intersection typing rules are nothing but the set-theoretic definition of intersection, and we immediately obtain for this identity the usual properties of set intersection:

idempotence:	$\sigma \wedge \sigma = \sigma$
commutativity:	$\sigma \wedge \tau = \tau \wedge \sigma$
associativity:	$(\sigma \wedge \tau) \wedge \rho = \sigma \wedge (\tau \wedge \rho)$
the universe is the neutral element:	$\sigma \wedge \omega = \sigma$.

In other words, there obviously exists a derivation of $\Gamma \vdash M{:}\,\sigma \wedge \sigma$ if and only if there is a derivation of $\Gamma \vdash M{:}\,\sigma$, etc.

In the following we will implicitly consider type expressions modulo the identity induced by the above equations, we will write as usual $\sigma\wedge\tau\wedge\rho$ without parentheses, we'll never write, since useless, $\sigma \wedge \sigma$ or $\sigma \wedge \omega$. Moreover, to spare parentheses, we assume that "$\wedge$" takes precedence over "$\to$".

To start perceiving the difference made by the introduction of the intersection, we must recall that, though in a type assignment system a term may be given a plurality of types, in the simple system without intersection it cannot have them "at the same time", in the sense that each (sub)term occurrence within another term may only be given one type, though different occurrences may be assigned different types. That formally connects with the fact that in the assumptions Γ there cannot be, by definition, distinct type assumptions for the same variable. For example, self-application cannot be typed, because to type the term xx one should assume for the variable x both a type σ and a type $\sigma \to \tau$.

With the intersection, on the other hand, the above constraint is not violated but circumvented: all is needed is to assume for x a type of the form $(\sigma \to \tau) \wedge \sigma$. Self-application $\lambda x.xx$ can thus be typed, through a successive arrow introduction:

$$\dfrac{\dfrac{\dfrac{x:(\sigma \to \tau) \wedge \sigma \vdash x:(\sigma \to \tau) \wedge \sigma}{x:(\sigma \to \tau) \wedge \sigma \vdash x:\sigma \to \tau}\,(\wedge\mathrm{E}) \qquad \dfrac{x:(\sigma \to \tau) \wedge \sigma \vdash x:(\sigma \to \tau) \wedge \sigma}{x:(\sigma \to \tau) \wedge \sigma \vdash x:\sigma}\,(\wedge\mathrm{E})}{x:(\sigma \to \tau) \wedge \sigma \vdash xx:\tau}\,(\to\mathrm{E})}{\vdash \lambda x.xx:(\sigma \to \tau) \wedge \sigma \to \tau}\,(\to\mathrm{I})$$

A completely different approach would consist in admitting recursive type definitions [22], and thus simply equating the two types: $\sigma = \sigma \to \tau$.

Adding the universal type ω not only trivially allows the typing of any term, even the term $\mathbf{\Omega}$, i.e., $(\lambda x.xx)(\lambda x.xx)$, which cannot be typed in any other way, but - owing to that - allows non-trivial (in a sense to be explained later) typings of terms that are not typable in the system with intersection and arrow only (or, worse, with arrow alone), for example the term $y\mathbf{\Omega}$, or the term $\lambda y.y\mathbf{\Omega}$:

$$\dfrac{\dfrac{y:\omega \to \sigma \vdash y:\omega \to \sigma\,(\mathrm{var}) \quad y:\omega \to \sigma \vdash \mathbf{\Omega}:\omega\,(\omega)}{y:\omega \to \sigma \vdash y\mathbf{\Omega}:\sigma}\,(\to\mathrm{E})}{\vdash \lambda y.y\mathbf{\Omega}:(\omega \to \sigma) \to \sigma}\,(\to\mathrm{I}) \tag{1}$$

The presence of the universal type also allows new non-trivial typings for terms already typable by intersection, since subterm typings may now be replaced by ω-typings: instead of being obliged to type every subterm, if the subterm is in argument position, we may choose to completely disregard its structure as if it were undefined, and simply assign it the type ω.

For example, for the term $\lambda x.xx$ we have the following additional type assignment derivations:

$$\dfrac{x:\sigma \vdash xx:\omega\,(\omega)}{\vdash \lambda x.xx:\sigma \to \omega}\,(\to\mathrm{I})$$

$$\dfrac{\dfrac{x:\omega \to \tau \vdash x:\omega \to \tau\,(\mathrm{var}) \quad x:\omega \to \tau \vdash x:\omega\,(\omega)}{x:\omega \to \tau \vdash xx:\tau}\,(\to\mathrm{E})}{\vdash \lambda x.xx:(\omega \to \tau) \to \tau}\,(\to\mathrm{I})$$

The fixed-point combinator $\mathbf{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ can also be typed, with an infinity of non-trivial types whose meaning will be examined again later. For two of such typings, namely:

$$\vdash \mathbf{Y}:(\omega \to \sigma_1) \to \sigma_1 \quad \text{and} \quad \vdash \mathbf{Y}:(\omega \to \sigma_1) \wedge (\sigma_1 \to \sigma_2) \to \sigma_2$$

we report the respective derivations, using here the natural-deduction notation without explicit bases, in order to keep the proof trees more manageable and easier to read. In this formalism the basis is just the set of undischarged assumptions; the only discharging rule is arrow introduction:

$$\dfrac{\begin{array}{c}[x:\sigma]^{(i)}\\ \vdots \\ M:\tau\end{array}}{\lambda x.M:\sigma \to \tau}\,(\to\mathrm{I})\,{}_{(i)}$$

Discharged assumptions are enclosed in square brackets, and labeled with numbers: these numbers are reported at $(\to\mathrm{I})$ inferences to indicate which assumptions those inferences actually discharge. Observe that here there is a slight mismatch with the previous calculus: strictly speaking we do not discharge assumptions out of a set, but occurrences of assumptions in the derivation tree. More: we are not forced neither to discharge all of them (i.e. all the occurrences of the premise $x{:}\sigma$ in the leaves of the derivation tree) nor at least one. This is reflected by the fact that weakening is admissible[2] in the former calculus, and that it is not required that x occurs in M.

The derivation of the first typing is:

$$\dfrac{\dfrac{\dfrac{\dfrac{[f:\omega \to \sigma_1]^{(1)} \quad xx:\omega\ (\omega)}{f(xx):\sigma_1}\,(\to\mathrm{E})}{\lambda x.f(xx):\omega \to \sigma_1}\,(\to\mathrm{I}) \qquad \lambda x.f(xx):\omega\ (\omega)}{(\lambda x.f(xx))(\lambda x.f(xx)):\sigma_1}\,(\to\mathrm{E})}{\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)):(\omega \to \sigma_1) \to \sigma_1}\,(\to\mathrm{I})\,{}_{(1)}$$

Below is the derivation of the second typing, where σ is $(\omega \to \sigma_1) \wedge (\sigma_1 \to \sigma_2)$, and where the omitted subderivation occurs in the previous tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[f:\sigma]^{(2)}}{f:\sigma_1 \to \sigma_2}\,(\wedge\mathrm{E}) \quad \dfrac{[x:\omega \to \sigma_1]^{(1)} \quad x:\omega}{xx:\sigma_1}\,(\to\mathrm{E})}{f(xx):\sigma_2}\,(\to\mathrm{E})}{\lambda x.f(xx):(\omega \to \sigma_1) \to \sigma_2}\,(\to\mathrm{I})\,{}_{(1)} \qquad \begin{array}{c}\dfrac{[f:\sigma]^{(2)}}{f:\omega \to \sigma_1}\,(\wedge\mathrm{E})\\ \vdots \\ \lambda x.f(xx):\omega \to \sigma_1\end{array}}{(\lambda x.f(xx))(\lambda x.f(xx)):\sigma_2}\,(\to\mathrm{E})}{\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)):(\omega \to \sigma_1) \wedge (\sigma_1 \to \sigma_2) \to \sigma_2}\,(\to\mathrm{I})\,{}_{(2)}$$

[2]A rule

$$\dfrac{A_1 \dots A_n}{B}\,\mathcal{R}$$

is *admissible* in a system $\mathcal{S}$ if, whenever (instances of) its premises $A_1, \dots, A_n$ are derivable in $\mathcal{S}$, also (the corresponding instance of) its conclusion B is derivable in $\mathcal{S}$. The precise statement of the admissibility of weakening is in the remarks following Definition 2.1.

In constructing typing derivations one may sometimes want to combine two independently built trees to obtain a new derivation: to this end the possibly different bases in (the leaves of) the two trees must be put together into a common one. This works well since weakening is an admissible rule in our systems, provided we consider (as we do) λ-terms modulo renaming of bound variables.

It is therefore useful to define the union $\Gamma \uplus \Gamma'$ of two bases Γ and Γ' as the basis consisting of the union set of Γ and Γ', with pairs $x{:}\,\sigma$, $x{:}\,\tau$ (of assumptions on the same variable in the two bases) replaced by $x{:}\,\sigma \wedge \tau$. More formally:

Definition 2.1 (Union of bases)

$$\begin{array}{rcl} \Gamma \uplus \Gamma' & = & \{x{:}\,\sigma \wedge \tau \mid x{:}\,\sigma \in \Gamma \textit{ and } x{:}\,\tau \in \Gamma'\} \\ & \cup & \{x{:}\,\sigma \mid x{:}\,\sigma \in \Gamma \textit{ and } x \textit{ does not occur in } \Gamma'\} \\ & \cup & \{x{:}\,\tau \mid x{:}\,\tau \in \Gamma' \textit{ and } x \textit{ does not occur in } \Gamma\}. \end{array}$$

Accordingly we have:

$$\Gamma \Subset \Gamma' \;\Leftrightarrow\; \exists \Gamma''.\, \Gamma \uplus \Gamma'' = \Gamma'.$$

Since terms are considered modulo α-conversion (i.e., renaming of bound variables), it is easy to verify, by induction on deductions, that weakening is admissible, more precisely that $\Gamma \vdash M{:}\,\tau$ and $\Gamma \Subset \Gamma'$ imply $\Gamma' \vdash M{:}\,\tau$.

In general, in a type assignment system the form of the term determines the forms of the types that can be assigned to that term, and possibly to its immediate subterms. The statement of such a property is traditionally called the *generation lemma* [16]; it consists of a sort of "reading backwards" of the typing rules, and it is essential for the study of the characteristics of a type system.
A proposition of this kind may also be proved for the system CDV [33].

Lemma 2.2 (Generation Lemma) *In the system CDV, considering types modulo idempotency, commutativity and associativity:*

1. $\Gamma \vdash x{:}\,\tau$ *and* $\tau \neq \omega$ *if and only if there exists* σ *such that* $x{:}\,\tau \wedge \sigma \in \Gamma$;
2. $\Gamma \vdash \lambda x.M{:}\,\tau$ *and* $\tau \neq \omega$ *if and only if there exist* $\sigma_1, \dots \sigma_n, \rho_1, \dots \rho_n$, *with* $n \geq 1$, *such that* $\tau = (\sigma_1 \to \rho_1) \wedge \cdots \wedge (\sigma_n \to \rho_n)$;
3. $\Gamma \vdash \lambda x.M{:}\,\sigma \to \tau$ *if and only if* $\Gamma, x{:}\,\sigma \vdash M{:}\,\tau$;
4. $\Gamma \vdash MN{:}\,\tau$ *and* $\tau \neq \omega$ *if and only if there exist* $\sigma_1, \dots \sigma_n, \tau_1, \dots \tau_n$, *with* $n \geq 1$, *such that* $\tau = \tau_1 \wedge \cdots \wedge \tau_n$, *and* $\Gamma \vdash M{:}\,\sigma_1 \to \tau_1$, $\Gamma \vdash N{:}\,\sigma_1$, $\dots$, $\Gamma \vdash M{:}\,\sigma_n \to \tau_n$, $\Gamma \vdash N{:}\,\sigma_n$.

3 Types and β-convertibility

In every well-behaved type system types are preserved by β-reduction, in the sense that if a term is typable with a certain type, it continues to be typable with that type after any number of reduction steps. In more concrete terms, in a strongly typed programming language, computation cannot change the type of an expression to a completely different and incompatible type, or restrict the set of types that can be assigned to it; for example, computation cannot transform an integer expression into a boolean, or transform an integer (and therefore also real) expression into a real which is not an integer. From a proof-theoretic point of view, that amounts to the fact that proof normalization does not modify the theorem being proved.

Actually, if we restrict ourselves to arrow types, the typing with τ of a term $(\lambda x.M)N$ may only have been obtained by arrow elimination from two typings respectively of the forms $\lambda x.M\colon \sigma \to \tau$ and $N\colon \sigma$, and in turn the typing of $\lambda x.M$ may only result by arrow introduction, i.e. by deriving the typing $M\colon \tau$ from the assumption $x\colon \sigma$. The typing with τ of the reduced term $M[N/x]$ is then derived by plugging a copy of the derivation of $N\colon \sigma$ into each occurrence of the assumption $x\colon \sigma$, as sketched in Fig.1.

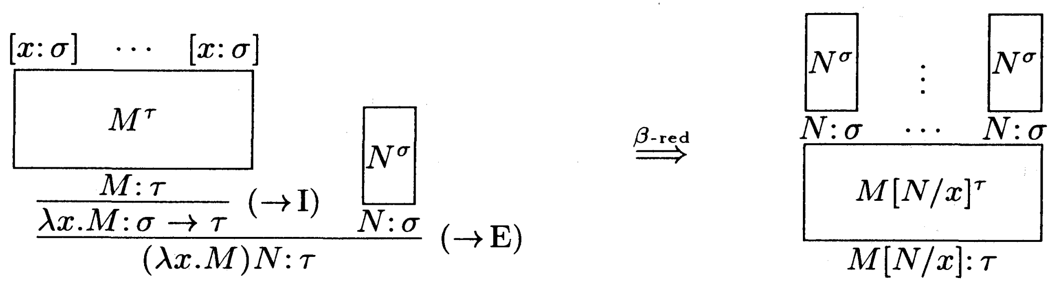


Figure 1: β-reduction

On the other hand, types are in general not preserved by β-expansion, and thus by β-convertibility: when computation is "reversed" from $M[N/x]$ to $(\lambda x.M)N$, the type can only be preserved by "unplugging" from M – at all the "sockets" corresponding to occurrences of the x variable – the type derivations of occurrences of N, and by substituting them with one type assumption for the x variable and one type derivation for N. This is surely feasible if M has exactly one occurrence of x; on the contrary, if M has multiple occurrences of x, the different typings of N cannot be unified unless they are identical. On the other hand, if in M there are no occurrences of x, there is no socket, and thus no type constraint on N, which can be anything, in particular a non-typable term. In other words, when the number of occurrences is different from one, either there are possibly too many types (i.e., more than one) for N, or no type for N.
Clearly, the first problem is solved by the intersection, which allows to give N (and x) the intersection of all the needed types, and the second problem is solved by the universal type, which allows to type any term [33].
Figure 2 gives a schematic view of the problem and the concerned proof transformations.

As a standard intuitive example of the first problem, consider – in a λ-calculus enriched with pairs and basic constant types, but the example could be coded

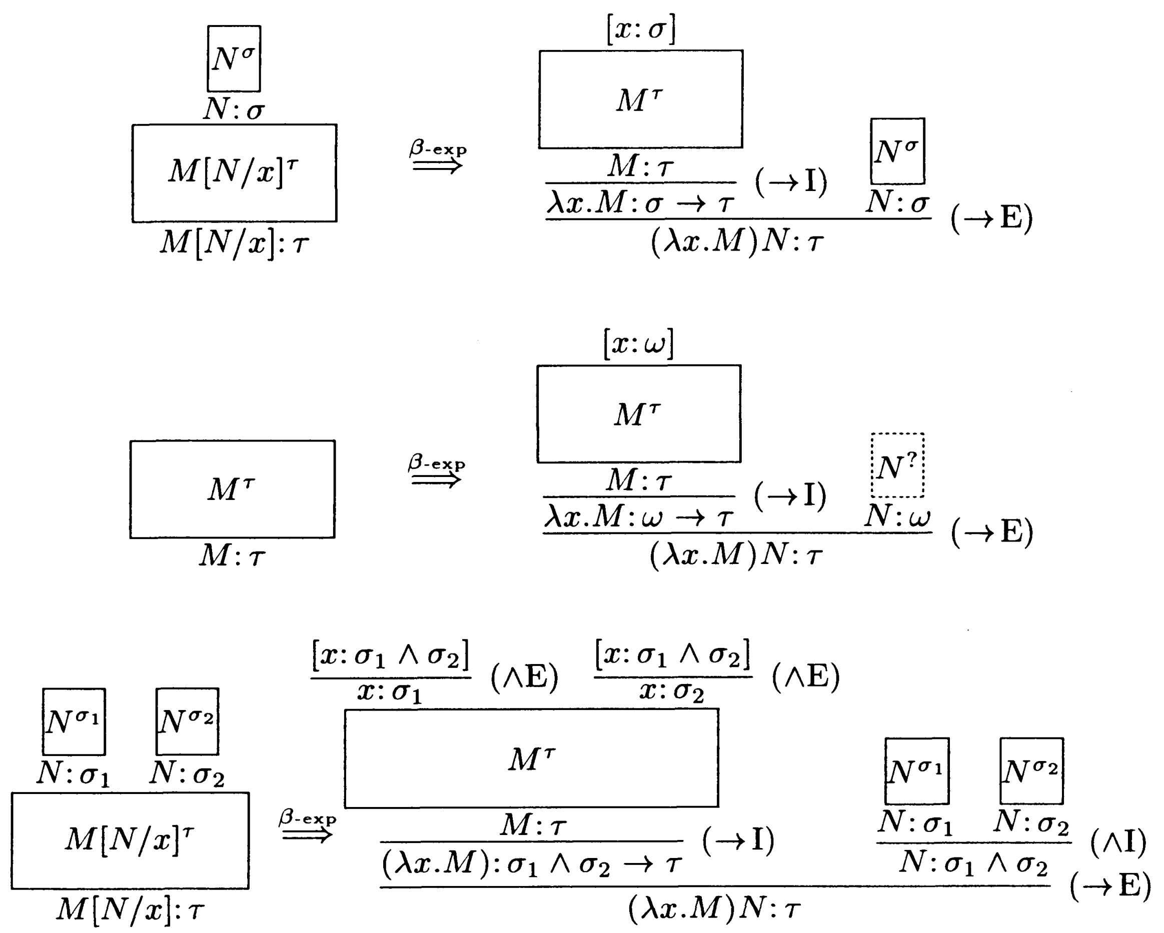


Figure 2: β-expansion

in pure calculus – the term consisting of the pair $\langle(\lambda x.x)3, (\lambda x.x)\ \mathsf{true}\rangle$. It has type $\mathsf{int} \times \mathsf{bool}$, since the first occurrence of the identity is given type $\mathsf{int} \to \mathsf{int}$, while the second occurrence is given type $\mathsf{bool} \to \mathsf{bool}$. The β-expanded term $(\lambda z.\langle z3, z\ \mathsf{true}\rangle)(\lambda x.x)$ cannot be typed in the simple system without intersection; with intersection, on the contrary, it is typable with the same type, since the identity may be assigned the type $(\mathsf{int} \to \mathsf{int}) \wedge (\mathsf{bool} \to \mathsf{bool})$, and the λ-abstraction the type $((\mathsf{int} \to \mathsf{int}) \wedge (\mathsf{bool} \to \mathsf{bool})) \to (\mathsf{int} \times \mathsf{bool})$.

The standard example of the second problem is the expansion from a typable term, such as the integer 3, to $(\lambda x.3)\mathbf{\Omega}$; or, to keep to the pure calculus, from $\lambda x.x$ to $(\lambda yx.x)\mathbf{\Omega}$. The expanded term, which obviously cannot be typed in the simple calculus, in the system with ω keeps the type of the reduced term, simply because $\mathbf{\Omega}$ can be assigned type ω, and $\lambda x.3$ the type $\omega \to \mathsf{int}$ – or $\lambda yx.x$ the type $\omega \to \varphi \to \varphi$.

Remark that the property of type preservation under β-expansion does not hold in the restricted system $\mathrm{CDV}_{\not\omega}$ (with intersection but without ω).

4 Unsolvable terms and typing

In any theory of computation an important role is played by the notion of approximation. One wants to formalize the idea that the result of a computation, i.e., the information finally returned as result, is not abruptly and magically created, but gradually built, step by step, by the computation itself (in contrast to the set-theoretic static notion of function as graph). A program may fail to terminate, but still go on building some information, though never completing it, as for example in a call of a function whose recursive definition were $f(x) = x \cdot f(x+1)$, or in a program building an infinite list, etc.

Moreover, since in λ-calculus recursion is simulated through the fixed-point combinator **Y**, and reduction does not stop when reaching an abstraction, the fixed-point combinator itself and all the terms corresponding to recursively defined functional values do not have normal forms, still they may compute (**Y** certainly does!) something interesting, because they go on repeatedly unfolding their recursive definitions, and when applied to proper arguments the application may reduce to a normal form.
This concept of partially computed result is formalized, within the λ-calculus, through the notions of *head normal form* and *approximant*. Recall that by definition a head normal form (*hnf*) is a term of the form $\lambda x_1 x_2 \ldots x_n.xM_1M_2\ldots M_m$ with $n, m \geq 0$, and where x is any variable, either free or identical to one of the x_i.

Terms having (i.e., reducible to) head normal forms, or *solvable terms*, intuitively are the terms which either have normal forms or, though nonterminating, compute something interesting in the above sense. Thus, the fixed-point combinator **Y** does not have normal form, but it has *hnf*, and so does, for example, the functional value corresponding to the recursive definition of factorial, and generally every term corresponding to a program that builds some information content.

On the other hand, terms without *hnf*, or *unsolvable terms*, represent nonterminating programs that do not produce any information content. The simplest unsolvable term is $\mathbf{\Omega}$, corresponding to the recursive definition $f = f$ (or also $f() = f()$), can only be assigned the type ω. Another example of an unsolvable term only typable by ω is $(\lambda x.xxx)(\lambda x.xxx)$.
The unsolvable term $\lambda z.\mathbf{\Omega}$, roughly corresponding to the function definition $g(z) = f$ *where* $f = f$, is immediately recognized to have, in addition to ω, any type $\sigma \to \omega$. More generally, a term of the form $\lambda x_1 \ldots x_n.\mathbf{\Omega}$ is typable with the types of the forms:

$$\omega, \quad \sigma_1 \to \omega, \quad \sigma_1 \to \sigma_2 \to \omega, \quad \ldots \quad \sigma_1 \to \sigma_2 \to \ldots \to \sigma_n \to \omega.$$

The natural n, which is the number of arguments the term has to be applied before returning $\mathbf{\Omega}$, is the *order of unsolvability*. The term **YK**, i.e., $\mathbf{Y}(\lambda fy.f)$ corresponding to a function whose recursive definition is $f(y) = f$, is an unsolvable term of infinite order, since it has types:

$$\omega, \quad \sigma_1 \to \omega, \quad \sigma_1 \to \sigma_2 \to \omega, \quad \cdots \quad \sigma_1 \to \sigma_2 \to \cdots \to \sigma_i \to \omega, \quad \cdots$$

for every natural i, as we may easily obtain, owing to the type invariance w.r.t. β-conversion, by typing each term of the infinite sequence:

$$\mathbf{YK} =_\beta \lambda y_1.\mathbf{YK} =_\beta \lambda y_1 y_2.\mathbf{YK} =_\beta \cdots =_\beta \lambda y_1 y_2 \ldots y_i.\mathbf{YK} =_\beta \cdots$$

It may be interesting to compare the above types of **YK** with a recursive type for the same term: from the recursive definition $f(y) = f$ we immediately obtain the equation $\tau = \sigma \to \tau$. In this case the above sequence of types looks like a sort of unfolding of the recursive definition; however, no property of this kind holds in general, and no significant relationship between the present system and recursive type systems is known.

5 Types and Approximants

The definition of head normal form, as we recalled in the previous section, allows formalizing the large distinction between computations that do or do not build an information content. The informal notion itself of (partial) information content or partially computed result is formalized by associating to every λ-term M another term $\alpha(M)$, which describes the structure of nested head normal forms of the term [63]. The mapping α could be inductively defined as follows.

$$\begin{array}{l}\alpha(\lambda x_1 \ldots x_n.xM_1 \ldots M_m) = \lambda x_1 \ldots x_n.x\,\alpha(M_1)\ldots\alpha(M_m)\\ \alpha(\lambda x_1 \ldots x_n.(\lambda x.P)Q\ M_1 \ldots M_m) = \lambda x_1 \ldots x_n.\mathbf{\Omega}\end{array}$$

where $n, m \geq 0$.

We prefer to think of the closed term $\mathbf{\Omega}$ as if it were a new primitive constant Ω denoting the "indefinite term", and of α as a mapping from ordinary λ-terms to *$\lambda\Omega$-terms*, where the set of $\lambda\Omega$-terms is defined by:

$$M ::= x|\Omega|\lambda x.M|MM.$$

$\lambda\Omega$-terms represent "partial terms", i.e. ordinary λ-terms where some subterms may be indefinite. We thus obtain the following definition of α.

Definition 5.1 (approximation mapping)

$$\begin{array}{l}\alpha(\lambda x_1 \ldots x_n.xM_1 \ldots M_m) = \lambda x_1 \ldots x_n.x\,\alpha(M_1)\ldots\alpha(M_m)\\ \alpha(\lambda x_1 \ldots x_n.(\lambda x.P)Q\ M_1 \ldots M_m) = \lambda x_1 \ldots x_n.\Omega\end{array}$$

where $n, m \geq 0$.

An *approximant* of a term M is a term $A \equiv \alpha(M')$ where M' is some term convertible to M. The set $\mathcal{A}(M)$ of all the approximants of a term M represents what informally is the set of all the possible partial results of computations of terms belonging to the same convertibility class as M.

A normalizable term has a finite number of distinct approximants. If, on the other hand, M does not have a normal form, $\mathcal{A}(M)$ may be finite or infinite, and constitutes a sort of generalized (possibly infinite) normal form. Thus, informally, if a program P goes on forever building as result an infinite list, its $\mathcal{A}(P)$ is the infinite list itself, represented by all its finite approximations.

In a sense, if we consider a term M, $\mathcal{A}(M)$ – being connected with convertibility - represents both "the past" and "the future" of the term, i.e., both the computation needed to build it in its "present" form and the (possibly infinite) result it will produce.

The type assignment system with intersection and ω may be extended to $\lambda\Omega$-terms without modifications, thus showing an immediate connection with the approximation structure of terms: if M is a term, the term $\alpha(M)$ precisely amounts to disregarding subterms corresponding to redexes still to be computed and thus giving them the type ω; conversely, typing with ω a non-redex subterm, as we did in Section 2, simply corresponds to typing a term $\alpha(M')$ for an M' obtained by expansion of the concerned subterm.

Then, if we take the types of all the approximants of a term, owing to type invariance w.r.t. conversion, we obtain all the possible types for the term itself. We thus have the following theorem, whose rather complex proof, based on the above intuition, may be found in [31, 44].

Theorem 5.2 (Approximation Theorem) *A term M is typable by a type σ w.r.t. a basis Γ if and only if there exists an approximant A of M that is typable by σ w.r.t. the same basis.*

The set of types assignable to a term thus perfectly reflects the approximation structure of the term.

For example, the set $\mathcal{A}(\lambda x.xx)$ of approximants of the term $\lambda x.xx$, which is a normal form, is the set of the $\lambda\Omega$-terms corresponding to all its possible β-expanded forms, which may be characterized through the use of the identity $\mathbf{I}(\equiv \lambda x.x)$:

1. an outermost redex, exemplified by $\mathbf{I}(\lambda x.xx)$, mapped into Ω;

2. an abstraction whose body is a redex, like the term $\lambda x.\mathbf{I}(xx)$, mapped to $\lambda x.\Omega$;

3. a *hnf* with an inner redex, like the term $\lambda x.x(\mathbf{I}x)$, mapped to $\lambda x.x\Omega$;

4. the term $\lambda x.xx$, mapped to itself.

We have therefore $\mathcal{A}(\lambda x.xx) = \{\Omega, \lambda x.\Omega, \lambda x.x\Omega, \lambda x.xx\}$; the typings of the four approximants are easily seen to be:

$$\vdash \Omega{:}\,\omega \quad \vdash \lambda x.\Omega{:}\,\sigma \to \omega \quad \vdash \lambda x.x\Omega{:}\,(\omega \to \tau) \to \tau \quad \vdash \lambda x.xx{:}\,(\sigma \to \tau) \wedge \sigma \to \tau.$$

No wonder they are exactly the four different typings we had derived in Section 2 for this term, owing to the obvious identity between the Ω occurrences here and the ω-typed subterms there.

The set of approximants of $\mathbf{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ is obtained from its infinite convertibility sequence:

$$\mathbf{IY} =_\beta \mathbf{Y} =_\beta \lambda f.f(\mathbf{Y}f) =_\beta \lambda f.f(f(\mathbf{Y}f)) =_\beta \dots$$

by applying the mapping α:

$$\mathcal{A}(\mathbf{Y}) = \{\Omega, \lambda f.\Omega, \lambda f.f\Omega, \lambda f.f(f\Omega), \dots\}.$$

As in the previous example, by typing the members of $\mathcal{A}(\mathbf{Y})$ we find again the types we had assigned to $\mathbf{Y}$ in Section 2:

$$\omega, \quad (\omega \to \sigma_1) \to \sigma_1, \quad (\omega \to \sigma_1) \wedge (\sigma_1 \to \sigma_2) \to \sigma_2, \quad \dots$$

in general $(\omega \to \sigma_1) \wedge (\sigma_1 \to \sigma_2) \wedge \cdots \wedge (\sigma_{i-1} \to \sigma_i) \to \sigma_i$ for every natural i. This sequence of typings represents a sort of precise typing of its "infinite normal form", in contrast with the typing it receives in recursive type systems, where the types making the sequence ω, σ_1, $\sigma_2, \dots \sigma_i, \dots$ are all identified and thus the sequence of intersection types for $\mathbf{Y}$ collapses in the simple finite type $(\sigma \to \sigma) \to \sigma$.

Analogously, consider again the term $\mathbf{YK}$ with its infinite convertibility sequence:

$$\mathbf{YK} =_\beta \lambda y_1.\mathbf{YK} =_\beta \lambda y_1 y_2.\mathbf{YK} =_\beta \cdots =_\beta \lambda y_1 y_2 \dots y_i.\mathbf{YK} =_\beta \cdots$$

The set of its approximants is then:

$$\mathcal{A}(\mathbf{YK}) = \{\Omega, \lambda y_1.\Omega, \lambda y_1 y_2.\Omega, \dots, \lambda y_1 \dots y_i.\Omega, \dots\}$$

which of course gives exactly the types found for $\mathbf{YK}$ in the preceding section.

6 The Characterization Theorems

As we anticipated in Section 1, type systems with intersection, with and without the universal type, are directly connected with the normalization properties of λ-terms, in the sense that the classes of solvable, normalizable, strongly normalizable terms, are exactly the classes of terms that may be assigned – in one system or the other – certain remarkable forms of types [33, 77].

Theorem 6.1 *A term M is (weakly) normalizable – i.e., it has a normal form – if and only if there are a basis Γ and a type σ, neither of them containing occurrences of ω, such that the typing $\Gamma \vdash M{:}\sigma$ is derivable in the system CDV.*

Of course ω, which cannot occur in the above typing judgment, may nevertheless occur in its derivation (which is performed in the system with ω).

The only-if part is easy to prove: if N is a normal form, the existence of an ω-free type assignment for N is proved by simple induction on term structure; the extension to all terms then follows by type invariance w.r.t. β-expansion.

The induction basis – i.e., the term in normal form is a variable – is obvious. As for the induction step, consider abstraction and application separately. The case of abstraction is straightforward: let N be $\lambda x.M$; by induction hypothesis a typing $\Gamma, x{:}\,\sigma \vdash M{:}\,\tau$ holds, where Γ, σ, τ do not contain occurrences of ω. Then, by arrow introduction, the typing $\Gamma \vdash \lambda x.M{:}\,\sigma \rightarrow \tau$ is immediately derived, where obviously ω does not occur in $\sigma \rightarrow \tau$.

The case of application M_1M requires, for the induction to work properly, a strengthening of the proposition to be proved: namely, an application (in normal form) is proved to be typable by a type variable.

Since M_1M is in normal form, the term M_1 cannot be an abstraction. If M_1 is a variable, and so the application is of the form xM, by induction hypothesis we have a derivation of $\Gamma, x{:}\,\sigma \vdash M{:}\,\tau$ for some Γ, σ, τ, with ω not occurring in Γ, σ, τ. If $\Gamma' \equiv \Gamma, x{:}\,\sigma \wedge (\tau \rightarrow \varphi)$, by weakening we get $\Gamma' \vdash M{:}\,\tau$. We may then build the following derivation:

$$\dfrac{\dfrac{\Gamma' \vdash x{:}\,\sigma \wedge (\tau \rightarrow \varphi) \quad (\text{var})}{\Gamma' \vdash x{:}\,\tau \rightarrow \varphi}\,(\wedge\text{E}) \qquad \Gamma' \vdash M{:}\,\tau}{\Gamma' \vdash xM{:}\,\varphi}\,(\rightarrow\text{E})$$

If M_1 is in turn an application, by induction hypothesis it may be typed by a type variable: $\Gamma_1 \vdash M_1{:}\,\psi$. Then by an obvious property we also have the typing $\Gamma_1[\tau/\psi] \vdash M_1{:}\,\tau$ for any type τ. Let now be $\Gamma_2 \vdash M{:}\,\sigma$ a typing for M; owing to the previous property we have the typing $\Gamma_1{}' \vdash M_1{:}\,\sigma \rightarrow \varphi$ where $\Gamma_1{}' \equiv \Gamma_1[(\sigma \rightarrow \varphi)/\psi]$. Taking the union $\Gamma \equiv \Gamma_1{}' \uplus \Gamma_2$ of the two bases, we obtain by arrow elimination the desired typing of M_1M:

$$\dfrac{\Gamma \vdash M_1{:}\,\sigma \rightarrow \varphi \quad \Gamma \vdash M{:}\,\sigma}{\Gamma \vdash M_1M{:}\,\varphi}\,(\rightarrow\text{E})$$

Observe that the above inductive argument actually proves the stronger result that every term in normal form may be typed in the type system without ω, i.e., for every normal form N there are a basis Γ and a type σ such that the conclusion $\Gamma \vdash N{:}\,\sigma$ may be derived without using ω in the derivation. This, however, does not mean that every term *having* normal form may be typed in the system without ω, since – as we remarked in Section 3 – types are not preserved under β-expansion, in absence of ω.

The if-part of the statement can be proved via a particularization of standard cut-elimination theorems[3]. This consists in progressively eliminating all the "detours" in a derivation, namely any introduction of a type constructor immediately followed by an elimination of the same constructor. As there are two such con-

[3]These have usually long and technical proofs. We give here only the structure of such a proof for the present case, adapting [39], Ch. 6, but any good text of proof theory includes cut-elimination, also called proof normalization. For a similar proof see [17].

structors we have two kinds of transformations over derivations:

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}_1\\ \Gamma \vdash M:\sigma_1\end{matrix} \quad \begin{matrix}\mathcal{D}_2\\ \Gamma\vdash M:\sigma_2\end{matrix}}{\Gamma\vdash M:\sigma_1\wedge\sigma_2}\,(\wedge\mathrm{I})}{\Gamma\vdash M:\sigma_i}\,(\wedge\mathrm{E}) \qquad \rightsquigarrow \qquad \begin{matrix}\mathcal{D}_i\\ \Gamma\vdash M:\sigma_i\end{matrix}$$

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}_1\\ \Gamma, x:\sigma \vdash M:\tau\end{matrix}}{\Gamma\vdash \lambda x.M:\sigma\to\tau}\,(\to\mathrm{I}) \quad \begin{matrix}\mathcal{D}_2\\ \Gamma\vdash N:\sigma\end{matrix}}{\Gamma\vdash (\lambda x.M)N:\tau}\,(\to\mathrm{E}) \qquad \rightsquigarrow \qquad \begin{matrix}\mathcal{D}_2\\ \Gamma\vdash N:\sigma\\ \mathcal{D}_1[N/x]\\ \Gamma\vdash M[N/x]:\tau\end{matrix}$$

where the right hand side of the second rule is the derivation obtained from $\mathcal{D}_1$ by replacing each occurrence of the axiom $\Gamma, x:\sigma \vdash x:\sigma$ by an occurrence of the deduction $\begin{matrix}\mathcal{D}_2\\ \Gamma\vdash N:\sigma\end{matrix}$, and each other judgment $\Gamma, x:\sigma \vdash P:\rho$ by $\Gamma \vdash P[N/x]:\rho$. Observe that this rule is exactly the transformation sketched in Figure 1.

The left hand side of these rules are *cuts*, namely detours which are eliminated by the application of the respective rule. The types $\sigma_1 \wedge \sigma_2$ and $\sigma \to \tau$ are the *cut types* of the respective cuts.

Once checked that these operations produce correct derivations, one has to show that there exists a procedure which reduces each deduction to a *normal deduction*, that is one which does not contain cuts. This is non trivial as reductions, in general, do not end up with shorter derivations: this is only true for reduction steps of the first rule, but it is surely false for the second one, which depends on the number of occurrences of the axioms $\Gamma, x:\sigma \vdash x:\sigma$ and on the size of the derivation $\mathcal{D}_2$.

The proof that any deduction is reducible, in a finite number of steps, to a normal one rests on the following facts. First we associate to each type a complexity measure called *rank*: $r(\varphi) = 0$ if $\varphi \in \mathcal{TV}$, $r(\sigma \to \tau) = r(\sigma \wedge \tau) = \max\{r(\sigma), r(\tau)\} + 1$. Second to each deduction $\mathcal{D}$ we associate a pair of natural numbers $cr(\mathcal{D}) = (d, n)$, the *cut rank* of $\mathcal{D}$, such that $d = \max\{r(\sigma) \mid \sigma \text{ is a cut type of } \mathcal{D}\}$, and n is the number of cuts whose cut types have maximal rank in $\mathcal{D}$ (shortly maximal cuts). These pairs are ordered by the lexicographic ordering

$$(d,n) \prec (d',n') \Leftrightarrow d < d' \text{ or } (d = d' \text{ and } n < n'),$$

which is a well-ordering (i.e. linear and without infinite descending chains).

The strategy to reach a normal proof consists in reducing maximal cuts such that all cuts above them have cut type of lower rank (but for this constrain, the choice of the maximal cut to be reduced is nondeterministic). One proves that if $\mathcal{D} \rightsquigarrow \mathcal{D}'$ in this way, then $cr(\mathcal{D}') \prec cr(\mathcal{D})$, since the elimination of a maximal cut which is not dominated by cuts of the same rank may only produce copies of cuts of lower rank, while the number of maximal cuts has been decreased by one. Iterating this process we eventually reach a derivation whose cut rank is $(0,0)$, which is a normal one.

The reduction process has effects on the subject M of its conclusion, say $\Gamma \vdash M:\sigma$. Indeed each reduction of a cut whose cut type is an arrow type corresponds to a contraction of a β-redex of M to its contractum. This implies

$$\overset{\mathcal{D}}{\Gamma \vdash M\colon \sigma} \overset{*}{\rightsquigarrow} \overset{\mathcal{D}'}{\Gamma \vdash N\colon \sigma} \;\Rightarrow\; M \overset{*}{\rightarrow}_\beta N.^4.$$

Therefore we are left to prove that if $\mathcal{D}'$ is a normal derivation such that ω occurs neither in σ nor in Γ, then N is a normal form (hence M is normalizable).

As is well-known, normal derivations satisfy the *subformula principle* (see e.g. [80], pp.41,42): if $\Gamma \vdash N\colon \sigma$ is the conclusion of the normal derivation $\mathcal{D}'$, then any type occurring in $\mathcal{D}'$ either occurs in σ or in some type in Γ. Therefore ω cannot occur in $\mathcal{D}'$, hence no inference in $\mathcal{D}'$ is an instance of rule (ω). One may wonder why we also ask that ω does not occur in Γ. Then consider the normal derivation:

$$\frac{x\colon \omega \to \varphi \vdash x\colon \omega \to \varphi \text{ (var)} \quad x\colon \omega \to \varphi \vdash M\colon \omega \ (\omega)}{x\colon \omega \to \varphi \vdash xM\colon \varphi} \ (\to\mathrm{E})$$

where M is any term (see also the typing example (1) of Section 2).

Using the particular structure of normal derivations one can show that the only way to construct a β-redex in N is to have introduction rules followed by elimination ones, contradicting the hypothesis on $\mathcal{D}'$.

Remark that only weak normalization holds: the original unreduced term may contain non-normalizable terms introduced via the (ω) rule. These are eliminated during the normalization process following the nondeterministic strategy established by the theorem or, for that matter, following any normalizing deterministic strategy (e.g. take always the leftmost maximal cut which is not dominated by cuts of the same rank). Infinite reduction sequences, of course, do not correspond to normalization of derivations, so that they only apply to redexes which are removed by the normalization procedure.

The characterization property concerning the solvable terms also intuitively follows from the same kind of argument.

Theorem 6.2 *A term M has a head normal form if and only if there are a basis Γ, an integer $n \geq 0$, and a type variable φ, such that (in the system CDV, with intersection and ω): $\Gamma \vdash M\colon \sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to \varphi$.*

Again, the only-if part is easy to prove. If a term M is in *hnf*:

$$M \equiv \lambda x_1 x_2 \ldots x_n . x M_1 M_2 \ldots M_m$$

^{4}As usual, $\overset{*}{\rightarrow}$ denotes the reflexive and transitive closure of $\rightarrow$.

a typing of the desired form may be obtained with the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\varDelta \vdash x{:}\,\omega^m \to \varphi \text{ (var)} \quad \varDelta \vdash M_1{:}\,\omega \;(\omega)}{\varDelta \vdash xM_1{:}\,\omega^{m-1} \to \varphi}\,(\to\mathrm{E})}{\vdots} \qquad \begin{array}{c}\\ \varDelta \vdash xM_1M_2 \ldots M_{m-1} : \omega \to \varphi \qquad \varDelta \vdash M_m{:}\,\omega \;(\omega)\end{array}}{\varDelta \vdash xM_1M_2 \ldots M_m : \varphi}\,(\to\mathrm{E})}{\vdots}$$

$$\dfrac{\varGamma, x_1{:}\,\sigma_1 \vdash \lambda x_2 \ldots x_n.xM_1M_2 \ldots M_m{:}\,\sigma_2 \to \cdots \to \sigma_n \to \varphi}{\varGamma \vdash \lambda x_1 \ldots x_n.xM_1M_2 \ldots M_m{:}\,\sigma_1 \to \cdots \to \sigma_n \to \varphi}\,(\to\mathrm{I})$$

where:

σ is $\omega^m \to \varphi \equiv \overbrace{\omega \to \omega \to \cdots \to \omega}^{m \text{ times}} \to \varphi$;	
$\varDelta$ is $\{x_1{:}\,\sigma_1, \ldots x_n{:}\,\sigma_n\}$ with $\sigma_i \equiv \sigma$,	if $x \equiv x_i$ for some $i \leq n$;
is $\{x{:}\,\sigma, x_1{:}\,\sigma_1, \ldots x_n{:}\,\sigma_n\}$	otherwise;
$\varGamma$ is empty	if $x \equiv x_i$ for some $i \leq n$;
is $\{x{:}\,\sigma\}$	otherwise.

As before, the invariance of typing by β-expansion ensures that any term reducible to an *hnf* may be assigned a type of that form.

The if-part may be viewed as a refinement of the analogous result for the characterization of normal forms. We will only sketch the argument here, referring to [17] and [32] for formal proofs. The type ω may now occur in $\varGamma$, $\sigma_1, \ldots \sigma_n$, but φ is not ω. Let again $\mathcal{D}$ be the derivation whose conclusion is the judgment:

$$\varGamma \vdash M{:}\,\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to \varphi.$$

As we have shown, $\mathcal{D}$ is reducible to a normal derivation $\mathcal{D}'$ whose conclusion is $\varGamma \vdash N{:}\,\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to \varphi$, for some N such that $M \overset{*}{\to}_\beta N$. Now $\mathcal{D}'$ can contain instances of rule (ω). The key observation is that a normal derivation can only assign type ω to β-redexes. In fact to give a type different from ω to a β-redex we need an application of rule ($\to$I) followed by an application of rule ($\to$E), i.e. a non-normal derivation. For the same reason a normal derivation can only assign type ω to applications of a β-redex to any number of arguments. Moreover if a term has only type ω, then its λ-abstractions can only have types of the shape $\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to \omega$, or intersections of types of this shape. We can conclude that N is a head normal form of M.

The third characterization property regards strongly normalizable terms.

Theorem 6.3 *A term M is strongly normalizable if and only if there are a basis $\varGamma$ and a type σ such that the typing judgment $\varGamma \vdash M{:}\,\sigma$ may be derived without using ω; i.e., if M may be typed – w.r.t. some basis $\varGamma$ – in the system $CDV_{\not\omega}$.*

For the only-if part, the previously shown typability of every normal form in the system without ω is not sufficient to prove the theorem, because of the failing of type invariance under β-expansion.

However, a weaker property may be proved, namely that if the contractum of a redex may be typed – w.r.t. some basis –, and also the argument in the redex may be typed – possibly w.r.t. a different basis –, then the whole redex may be assigned the same type as the contractum, w.r.t. the basis consisting of the *union* of the two bases.

From that property a lemma may be derived by structural induction on terms, namely that if a term may be typed, any term obtained from it by *strong leftmost expansion* may also be typed, possibly with a different type and w.r.t. a different basis; where by strong leftmost expansion we mean an expansion generating a leftmost redex whose argument may be typed. Finally, the (only-if part of) the theorem is proved from the lemma, by induction on the maximum of the lengths of the reduction paths [7] (Section 3.5).

The proof of the if-part is of the same kind as the proof of the previously mentioned strong normalization theorem of logic calculi, using the reducibility method due originally to Tait [89], with a double induction on types and derivations [77].

7 Subtyping

The notion of subtype, which has become popular in the last decades as one of the essential ingredients of object-oriented programming, was absent in the classical type systems for λ-calculus, as it was absent – at least as a general notion – in traditional typed programming languages like Pascal or C. As for functional programming, the language ML [67, 68, 71], well-known for its elegance and its sound mathematical foundation, just because of its neat type discipline had to give up even the traditional type inclusion between integers and reals.

In type systems with intersection, on the other hand, a notion of subtype naturally arises. If we identify, as we did in Section 2, each type with the set of terms having that type, we immediately have a notion of subtype as set-inclusion: we will write $\sigma \leq \tau$ if and only if for every basis Γ and every term M one has that $\Gamma \vdash M{:}\,\sigma$ implies $\Gamma \vdash M{:}\,\tau$. The relation $\leq$ is of course a partial order, satisfying the usual properties of set-inclusion:

reflexivity:	$\sigma \leq \sigma$
transitivity:	$\sigma \leq \rho \ \& \ \rho \leq \tau \quad \Rightarrow \quad \sigma \leq \tau$
inters. inclusion:	$\sigma \wedge \tau \leq \sigma \qquad \sigma \wedge \tau \leq \tau$
inters. monotonicity:	$\sigma_1 \leq \sigma_2 \ \& \ \tau_1 \leq \tau_2 \quad \Rightarrow \quad \sigma_1 \wedge \tau_1 \leq \sigma_2 \wedge \tau_2.$

Moreover, in the systems with the universal type, we have of course:

universe: $\sigma \leq \omega.$

Set-theoretic equality $\sigma = \tau$ is equivalent to $\sigma \leq \tau$ and $\tau \leq \sigma$.

We may consider the above properties as axioms and inference rules of a formal theory of the subtyping relation, which allows to derive propositions of the form $\sigma \leq \tau$. Then, instead of starting by identifying types modulo *idempotency*, *commutativity* and *associativity*, as in Section 2, we may obtain the same effect by explicitly adding idempotency as an axiom:

idempotency: $\sigma \leq \sigma \wedge \sigma$

and then considering equality between types as a defined relation:

$$\sigma = \tau \overset{\text{def}}{\Leftrightarrow} \sigma \leq \tau \;\&\; \tau \leq \sigma.$$

So, putting together the subtype rules (and axioms) with the typing rules for terms, we are again with a purely formal system, where we may perform derivations and whose properties we can study without reference to any interpretation. In order to really connect the two formal systems, the set-theoretic definition of subtype from which we started:

$$\sigma \leq \tau \quad \overset{\text{def}}{\Leftrightarrow} \quad \forall \Gamma, M.(\Gamma \vdash M{:}\,\sigma \;\Rightarrow\; \Gamma \vdash M{:}\,\tau) \tag{2}$$

must be read as a new typing rule, the ***subsumption rule***, which allows the propositions proved in the subtype theory to be employed to assign types to λ-terms:

$$\frac{\Gamma \vdash M{:}\,\sigma \quad \sigma \leq \tau}{\Gamma \vdash M{:}\,\tau}\;(\text{sub})$$

Notice that our subsumption rule uses the inclusion between types as axiomatized in an effective way at the beginning of the present section. By the way it is easy to check that the definition given in (2) is equivalent. The subsumption rule, along with the theory of the subtyping relation, dispenses with the intersection-elimination rules, which become derived rules:

$$\frac{\Gamma \vdash M{:}\,\sigma \wedge \tau \quad \sigma \wedge \tau \leq \sigma}{\Gamma \vdash M{:}\,\sigma}\;(\text{sub}) \qquad \frac{\Gamma \vdash M{:}\,\sigma \wedge \tau \quad \sigma \wedge \tau \leq \tau}{\Gamma \vdash M{:}\,\tau}\;(\text{sub})$$

More precisely, the type system CDV, with intersection introduction *and elimination*, is equivalent to the system $\text{CDV}_{\leq}$, reported in Table 2, where the elimination rules have been replaced by the subsumption rule and the subtype rules; among these the axiom $\sigma \leq \omega$ becomes superfluous and might be omitted, since the typing axiom $\Gamma \vdash M{:}\,\omega$ is already stronger (we keep it as it will again be necessary in the expanded systems considered in the next sections).

Analogously, the restricted system $\text{CDV}_{\not\omega}$ is equivalent to the system $\text{CDV}_{\leq,\not\omega}$, which of course is the system $\text{CDV}_{\leq}$ with the two resp. typing and subtype axioms for ω removed.

8 Subtyping and η-rule

The notion of function modeled by λ-calculus is, as we recalled, intensional: two functions that, when applied to the same argument always give the same result, are

The Theory of the Subtype Relation

$$\sigma \leq \sigma \text{ (refl)} \qquad \sigma \wedge \tau \leq \sigma \text{ (incl)} \qquad \sigma \wedge \tau \leq \tau \text{ (incl)} \qquad \sigma \leq \sigma \wedge \sigma \text{ (idem)}$$

$$\frac{\sigma \leq \rho \quad \rho \leq \tau}{\sigma \leq \tau} \text{ (trans)} \qquad \frac{\sigma_1 \leq \sigma_2 \quad \tau_1 \leq \tau_2}{\sigma_1 \wedge \tau_1 \leq \sigma_2 \wedge \tau_2} \text{ (mon)} \qquad \sigma \leq \omega \text{ (ω-incl)}$$

The Type Assignment System

$$\Gamma, x{:}\,\sigma \vdash x{:}\,\sigma \text{ (var)} \qquad \frac{\Gamma \vdash M{:}\,\sigma \to \tau \quad \Gamma \vdash N{:}\,\sigma}{\Gamma \vdash M{:}\,\tau} (\to\mathrm{E}) \qquad \frac{\Gamma, x{:}\,\sigma \vdash N{:}\,\tau}{\Gamma \vdash \lambda x.M{:}\,\sigma \to \tau} (\to\mathrm{I})$$

$$\frac{\Gamma \vdash M{:}\,\sigma \quad \Gamma \vdash M{:}\,\tau}{\Gamma \vdash M{:}\,\sigma \wedge \tau} (\wedge\mathrm{I}) \qquad \frac{\Gamma \vdash M{:}\,\sigma \quad \sigma \leq \tau}{\Gamma \vdash M{:}\,\tau} \text{ (sub)} \qquad \Gamma \vdash M{:}\,\omega \quad (\omega)$$

Table 2: The system CDV$_\leq$

not necessarily identical, since they may consist of different algorithms, in contrast to the set-theoretic extensional notion where functions are identified with their graphs. Extensionality in the untyped calculus may be recovered, as is well known, through the introduction of the η-reduction rule:

$$\lambda x.Mx \to M \quad \text{if} \quad x \notin \mathrm{FV}(M).$$

With the subsequent notion of $\beta\eta$-convertibility one obtains that any two functions being extensionally equal are identified.

A type system for the λ-calculus with the η-rule must also contain an η-typing-rule, in order to satisfy the minimal requirement of type invariance by η-reduction:

$$\frac{\Gamma \vdash \lambda x.Mx{:}\,\sigma \quad x \notin \mathrm{FV}(M)}{\Gamma \vdash M{:}\,\sigma} (\eta)$$

This does not mean that the resulting type system is extensional, since types are not preserved by η-expansion, as it can be easily recognized. For example, a variable may obviously be typed with a type variable: $x{:}\,\varphi \vdash x{:}\,\varphi$; if now we expand x to $\lambda y.xy$, the expanded term may no longer be assigned the atomic type φ, since by the Generation Lemma its type must be an arrow-type, as for instance in the following derivation:

$$\cfrac{\cfrac{x{:}\,\varphi \to \psi,\ y{:}\,\varphi \vdash x{:}\,\varphi \to \psi \text{ (var)} \quad x{:}\,\varphi \to \psi,\ y{:}\,\varphi \vdash y{:}\,\varphi \text{ (var)}}{x{:}\,\varphi \to \psi,\ y{:}\,\varphi \vdash xy{:}\,\psi} (\to\mathrm{E})}{x{:}\,\varphi \to \psi \vdash \lambda y.xy{:}\,\varphi \to \psi} (\to\mathrm{I})$$

The system CDV$_{\eta,\not\omega}$, consisting of the system CDV$_{\not\omega}$ augmented with the η-typing-rule, is nevertheless interesting in its own right, as we will immediately see. It allows λ-terms to be assigned more types than by the system without the η-rule;

for example, it allows the identity to be assigned a type having a shape different from $\sigma \to \sigma$, as shown in the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{z{:}\,\sigma \wedge \rho, x{:}\,\sigma \to \tau \vdash x{:}\,\sigma \to \tau \text{ (var)} \qquad \dfrac{z{:}\,\sigma \wedge \rho, x{:}\,\sigma \to \tau \vdash z{:}\,\sigma \wedge \rho \text{ (var)}}{z{:}\,\sigma \wedge \rho, x{:}\,\sigma \to \tau \vdash z{:}\,\sigma}(\wedge\text{E})}{z{:}\,\sigma \wedge \rho, x{:}\,\sigma \to \tau \vdash xz{:}\,\tau}(\to\text{E})}{x{:}\,\sigma \to \tau \vdash \lambda z.xz{:}\,\sigma \wedge \rho \to \tau}(\to\text{I})}{x{:}\,\sigma \to \tau \vdash x{:}\,\sigma \wedge \rho \to \tau}(\eta)}{\vdash \lambda x.x{:}\,(\sigma \to \tau) \to \sigma \wedge \rho \to \tau}(\to\text{I})$$

Now, observe that the axioms and rules of the subtype theory of the system $\text{CDV}_{\leq,\not\omega}$ do not in any way describe the fact that the arrow-type is the function type, and that therefore some (or all) elements of the universe are functions: they only axiomatize the usual set-theoretic inclusion, without any commitment to the nature of elements. Interestingly, the same effect as the η-typing-rule, i.e. type invariance by η-reduction, may alternatively be obtained by adding to the subtype theory two rules (actually, an axiom and a rule) that describe natural properties of sets of functions [17, 53]. This is clear when we interpret $\rho \to \sigma$ as stating that the function applied to an input having property ρ gives an output having property σ, not when we interpret $\rho \to \sigma$ as stating that the domain of the function is the set of elements with property ρ and the codomain of the function is the set of elements with property σ.

$$\begin{array}{rcll} (\rho \to \sigma) \wedge (\rho \to \tau) & \leq & \rho \to \sigma \wedge \tau & (\to\wedge) \\ \sigma' \leq \sigma \ \& \ \tau \leq \tau' & \Rightarrow & \sigma \to \tau \leq \sigma' \to \tau' & (\to\leq) \end{array}$$

The second formula $(\to\leq)$ expresses the well-known fact that functional types are covariant in the result type but contravariant in the argument type. The first formula $(\to\wedge)$ is actually an equality, since the opposite inequality may be derived from the formula $(\to\leq)$ and the previous axioms and rules:

$$\dfrac{\dfrac{\rho \leq \rho \text{ (refl)} \quad \sigma \wedge \tau \leq \sigma \text{ (incl)}}{\rho \to \sigma \wedge \tau \ \leq \ \rho \to \sigma}(\to\leq) \qquad \dfrac{\rho \leq \rho \text{ (refl)} \quad \sigma \wedge \tau \leq \tau \text{ (incl)}}{\rho \to \sigma \wedge \tau \ \leq \ \rho \to \tau}(\to\leq)}{\rho \to \sigma \wedge \tau \ \leq \ (\rho \to \sigma) \wedge (\rho \to \tau)}$$

where the last conclusion is deduced from the premises using (mon), (idem), and (trans). The new system thus resulting, i.e., the system $\text{CDV}_{\leq,\not\omega}$ plus the two new subtype rules, being a restriction of the system BCD we are going to introduce presently, will be indicated in the following as the system $\text{BCD}_{\not\omega}$; it is equivalent to the system $\text{CDV}_{\eta,\not\omega}$ (the original intersection system without ω, but augmented with η):

$$\text{BCD}_{\not\omega} \ \equiv \ \text{CDV}_{\leq,\not\omega} + (\to\wedge) + (\to\leq) \text{ is equivalent to } \text{CDV}_{\eta,\not\omega}.$$

If we leave implicit, as is usually done, the deduction steps performed in the theory of the subtype relation, the proof trees in $\text{CDV}_{\leq,\not\omega}$ are generally simplified w.r.t. their corresponding in $\mathrm{CDV}_{\eta,\not\omega}$. For example, the above typing derivation for the identity becomes:

$$\dfrac{\dfrac{x{:}\,\sigma \to \tau \vdash x{:}\,\sigma \to \tau \text{ (var)}}{\vdash \lambda x.x{:}\,(\sigma \to \tau) \to \sigma \to \tau}\,(\to\mathrm{I})}{\vdash \lambda x.x{:}\,(\sigma \to \tau) \to \sigma \wedge \rho \to \tau}\,(\mathrm{sub})$$

To obtain the equivalent (through subtyping) of the system CDV_η consisting of the complete CDV (i.e., with ω) plus the η-rule, one more axiom for the subtype relation is needed (besides the usual typing rule for ω), for consider again the term $\lambda y.xy$: it may be typed with $\omega \to \omega$ w.r.t. the empty basis, while with an empty basis its η-reduct x can only be assigned the type ω, but no arrow type. Type invariance by reduction thus requires the axiom

$$\omega \leq \omega \to \omega \quad (\omega\text{-}\eta)$$

which of course implies $\omega = \omega \to \omega$ (from $\sigma \leq \omega$).

Intuitively, the (ω-η) axiom states that every λ-term, even an atomic one (i.e., a variable), may be viewed as a function, which amounts to saying that every element of the universe is a function, at least of the generic functional type $\omega \to \omega$. The system we have finally obtained, reported in Table 3, equivalent to CDV_η, is known in the literature as the system BCD [17], while the previous system $\mathrm{BCD}_{\not\omega}$ simply is the system BCD with all the (typing and subtype) axioms for ω removed. Remark that the axiom $\omega = \omega \to \omega$, implying that every term is a function, does not imply that the system is extensional; on the contrary, just because of the functional character of all terms, the variable x and the abstraction $\lambda y.xy$ must be identified, which requires, as we observed, that the two terms have exactly the same types. This can only be achieved with some axiom stating that every type is equal to a functional type or to an intersection of functional types (see the end of Section 12). A simple axiom of that kind would be, for example, the equality $\varphi = \omega \to \varphi$.

The above discussion suggests that different λ-theories may be simply obtained by augmenting or modifying the theory of the subtype relation (without modifying the typing rules proper): this fact will reveal of paramount importance in the characterization of λ-models.

9 The System BCD

The system BCD is, as we pointed out, equivalent to the system CDV_η. Observe that in the system CDV_η there is no primitive notion of subtyping and thus of equality (other than syntactical coincidence) between types. As we showed, these relations may be introduced as derived notions, maintaining the point of view already adopted in CDV for commutativity, associativity, etc.

Nevertheless, also in view of the fact that the subtype relation is the basis for the semantics and it is therefore more fundamental, the opposite attitude, embodied

The Theory of the Subtype Relation

$$\sigma \leq \sigma \text{ (refl)} \qquad \sigma \wedge \tau \leq \sigma \text{ (incl)} \qquad \sigma \wedge \tau \leq \tau \text{ (incl)} \qquad \sigma \leq \sigma \wedge \sigma \text{ (idem)}$$

$$\frac{\sigma \leq \rho \quad \rho \leq \tau}{\sigma \leq \tau}\text{ (trans)} \qquad \frac{\sigma_1 \leq \sigma_2 \quad \tau_1 \leq \tau_2}{\sigma_1 \wedge \tau_1 \leq \sigma_2 \wedge \tau_2}\text{ (mon)}$$

$$(\rho \to \sigma) \wedge (\rho \to \tau) \leq \rho \to \sigma \wedge \tau \quad (\to\wedge)$$

$$\frac{\sigma' \leq \sigma \quad \tau \leq \tau'}{(\sigma \to \tau) \leq (\sigma' \leq \tau')}\ (\to\leq)$$

$$\sigma \leq \omega\ (\omega\text{-incl}) \qquad \omega \leq \omega \to \omega\ (\omega\text{-}\eta)$$

The Type Assignment System

$$\Gamma, x{:}\sigma \vdash x{:}\sigma \text{ (var)} \qquad \frac{\Gamma \vdash M{:}\sigma \to \tau \quad \Gamma \vdash N{:}\sigma}{\Gamma \vdash M{:}\tau}\ (\to\text{E}) \qquad \frac{\Gamma, x{:}\sigma \vdash N{:}\tau}{\Gamma \vdash \lambda x.M{:}\sigma \to \tau}\ (\to\text{I})$$

$$\frac{\Gamma \vdash M{:}\sigma \quad \Gamma \vdash M{:}\tau}{\Gamma \vdash M{:}\sigma \wedge \tau}\ (\wedge\text{I}) \qquad \frac{\Gamma \vdash M{:}\sigma \quad \sigma \leq \tau}{\Gamma \vdash M{:}\tau}\text{ (sub)} \qquad \Gamma \vdash M{:}\omega \quad (\omega)$$

Table 3: The System BCD

in the system BCD, is preferable: to axiomatize the partial order on types as a primitive notion, so that deductions based on subtyping really are derivations in the formal system and not metadeductions, and to have the intersection-elimination rules as (trivial and therefore scarcely interesting) derived rules.

Historically, the system BCD is the one that was originally proposed to account for the η-rule, while the system CDV$_\eta$ was only introduced for comparison with the simple Curry system [17].

As in the system with subtyping but without ω, typing derivations in BCD are easier to understand than their counterparts in CDV$_\eta$, simply because deductions in the subtype theory may be considered separately, and possibly left implicit in the actual writing.

An important feature of the system BCD (or of the system CDV$_\eta$ with the derived type equality), which leads to a further simplification of the whole theory, is the fact that it identifies with ω every type of the form $\sigma \to \omega$, as may easily be seen: from the axiom of universal inclusion $\sigma \leq \omega$ one has by contravariance $\omega \to \omega \leq \sigma \to \omega$, and so (owing to the equality $\omega \to \omega = \omega$) the inclusion $\omega \leq \sigma \to \omega$; of course the opposite inequality holds, hence the conclusion.

The characterization of solvable terms thus becomes the statement that they are exactly the terms that may be assigned a type different from ω, or – equivalently – that unsolvable terms are exactly the terms whose only type is the universal type ω. In conclusion, we have [17, 77]:

1. *solvable terms* are exactly the terms that may be typed with a type *different from* ω;

2. *normalizable terms* are exactly the terms that may be typed with a type *not containing* ω;

3. *strongly normalizable* terms are exactly the terms that may be typed *without using* ω *in the derivation.*

A bit more precisely, but using the symbol $\not\in$ sloppily as synonymous of "does not occur in":

Theorem 9.1 (BCD Characterization Theorem)
If M is a λ-term, then the following double implications hold:

1. *M is solvable (has hnf)* $\Leftrightarrow$ $\exists \Gamma,\, \sigma \neq_{\mathrm{BCD}} \omega \,.\, \Gamma \vdash_{\mathrm{BCD}} M{:}\,\sigma$

2. *M is normalizable (has nf)* $\Leftrightarrow$ $\exists \Gamma, \sigma \,.\, (\omega \not\in \Gamma, \sigma)\ \&\ \Gamma \vdash_{\mathrm{BCD}} M{:}\,\sigma$

3. *M is strongly normalizable* $\Leftrightarrow$ $\exists \Gamma, \sigma \,.\, \Gamma \vdash_{\mathrm{BCD}_{\not\omega}} M{:}\,\sigma$.

In the statement of the theorem we could have of course used, instead of the derivation relation $\vdash_{\mathrm{BCD}}$, the perfectly equivalent relation $\vdash_{\mathrm{CDV}_\eta}$; more concisely, we will indicate by the symbol $\vdash_\eta$ either of the two equivalent derivation relations, and will generally label by the subscript η the symbols of the related notions.

The notion of approximant is also simplified, since it must now be relative to $\beta\eta$-reduction. The mapping α from λ-terms to λΩ-terms is thus replaced by a mapping α_η which, taking into account the η-reduction, maps $\lambda x.\Omega$ directly to Ω [15](Definition 14.3.6(ii)).

Definition 9.2 (η-approximation mapping)

$$\begin{array}{l}\alpha_\eta(\lambda x_1 \ldots x_n.x M_1 \ldots M_m) = \lambda x_1 \ldots x_n.x\ \alpha_\eta(M_1) \ldots \alpha_\eta(M_m)\\ \alpha_\eta(\lambda x_1 \ldots x_n.(\lambda x.P)Q\ M_1 \ldots M_m) = \Omega\end{array}$$

where $n, m \geq 0$.

The set of η-approximants of a term is consequently defined as:

Definition 9.3 (η-approximants)

$$\mathcal{A}_\eta(M) \;=\; \{A \in \lambda\Omega\text{-}terms \mid \exists M' =_\beta M \,.\, A \equiv \alpha_\eta(M')\}.$$

We say that a λΩ-term A is an η-approximant *if and only if $A \equiv \alpha_\eta(M)$ for some λ-term M.*

The approximation theorem continues to hold in the system BCD, with the new notion of approximant [86].

Theorem 9.4 (BCD Approximation Theorem) *A term M is typable in BCD (or in CDV_η) by a type σ w.r.t. a basis Γ if and only if there exists an η-approximant A of M that is typable in BCD (or in CDV_η) by σ w.r.t. the same basis. In formula:*

$$\Gamma \vdash_\eta M{:}\sigma \quad \Leftrightarrow \quad \exists A \in \mathcal{A}_\eta(M) \,.\, \Gamma \vdash_\eta A{:}\sigma.$$

Finally, the Generation Lemma, which will be used in the next sections, is also simpler than the same lemma for the system CDV (Lemma 2.2).

Lemma 9.5 (BCD Generation Lemma) *In the system BCD:*

1. $\Gamma \vdash_\eta x{:}\tau \;\&\; \tau \neq_\eta \omega \quad \Leftrightarrow \quad \exists \sigma \,.\, x{:}\sigma \in \Gamma \;\&\; \sigma \leq_\eta \tau$;
2. $\Gamma \vdash_\eta \lambda x.M{:}\tau \quad \Leftrightarrow \quad \exists I, \sigma_i, \rho_i \,.\, \tau = \bigwedge_{i\in I}(\sigma_i \to \rho_i)$;
3. $\Gamma \vdash_\eta \lambda x.M{:}\sigma \to \tau \quad \Leftrightarrow \quad \Gamma, x{:}\sigma \vdash_\eta M{:}\tau$;
4. $\Gamma \vdash_\eta MN{:}\tau \quad \Leftrightarrow \quad \exists \sigma \,.\, \Gamma \vdash_\eta M{:}\sigma \to \tau \;\&\; \Gamma \vdash_\eta N{:}\sigma$.

The proof of Lemma 9.5, as performed in [17], is based on the following property of the type inclusion $\leq_\eta$ (defined in Table 8), which will also be used in Sections 10 and 13.

Proposition 9.6

$$\bigwedge_{i\in I}(\sigma_i \to \tau_i) \leq_\eta \sigma \to \tau \Rightarrow \bigwedge\{\tau_i \mid i \in I, \sigma \leq_\eta \sigma_i\} \leq_\eta \tau.$$

10 Semantics of Type Assignments

There are several ways of interpreting types in the case of typed λ-calculus: what is common to all of them is the fact that each term has exactly one type, determined by the term structure.

As we saw in the previous sections, Curry's approach assigns types to untyped terms, so that if a term is typable, then it has infinitely many types. In intersection type systems it is in general undecidable whether a term has a certain type, since, as we have seen, types characterize non computable properties of terms. The consequence we draw is that types, in the case we are interested in, cannot be interpreted as constraints for term formation. The basic intuition behind them is, instead, that they express properties of terms. From a semantical point of view λ-terms have their denotations in some λ-model, hence types will be associated to subsets of the carrier of this model.

We limit ourselves to considering λ-models in the category of ω-algebraic complete lattices, which is the relevant case here. Therefore, restricting the general definition (see [15], Section 5.4), we say that a *λ-model* is a triple $\mathcal{D} = \langle D, \Phi, \Psi\rangle$ where D is an ω-algebraic complete lattice, $[D \to D]$ is the space of Scott continuous functions from D to D, Φ and Ψ are morphisms such that:

1. $\Phi : D \to [D \to D]$,
2. $\Psi : [D \to D] \to D$,
3. $\Phi \circ \Psi = Id_{[D \to D]}$.

The existence of Φ determines an applicative structure over D, i.e., it induces the binary operation $\_ \cdot \_$ defined by

$$d \cdot e = \Phi(d)(e).$$

Because of Φ any element of D represents a function in $[D \to D]$: the elements of the image $\Phi[D]$ are called the *representable functions.* Vice versa, each function f in $[D \to D]$ has a representative $\Psi(f)$ in D. Then, if we denote by ξ a mapping from the set Var of term variables to D (a *term environment*), an interpretation map $[\![M]\!]^{\mathcal{D}}_{\xi} \in D$, which interprets λ-terms into the carrier D, can be defined as follows:

1. $[\![x]\!]^{\mathcal{D}}_{\xi} = \xi(x)$,
2. $[\![MN]\!]^{\mathcal{D}}_{\xi} = \Phi([\![M]\!]^{\mathcal{D}}_{\xi})([\![N]\!]^{\mathcal{D}}_{\xi}) = [\![M]\!]^{\mathcal{D}}_{\xi} \cdot [\![N]\!]^{\mathcal{D}}_{\xi}$,
3. $[\![\lambda x.M]\!]^{\mathcal{D}}_{\xi} = \Psi(f)$, where $f(d) = [\![M]\!]^{\mathcal{D}}_{\xi[x:=d]}$.

Here $\xi[x := d](y) = d$ if $x \equiv y$ (x and y are the same variable), $\xi(y)$ else. This definition is a good one since the function f of the third clause is in the domain of Ψ.

Once terms have been interpreted as elements of D, types denote properties of elements of D, which are naturally identified with their extensions, i.e. subsets of D (or, equivalently, elements of the powerset $\mathcal{P}(D)$ of D). If M is a closed term, the judgment $\vdash M{:}\,\sigma$ can be read as stating that the denotation of M in D under any environment ξ is in the interpretation of σ. Formally, if $[\![\sigma]\!]^{\mathcal{D}} \subseteq D$ is the interpretation of σ, then

$$\models_{\mathcal{D},\xi} M{:}\,\sigma \Leftrightarrow [\![M]\!]^{\mathcal{D}}_{\xi} \in [\![\sigma]\!]^{\mathcal{D}}.$$

In the systems of type assignment we derive statements of the form $\Gamma \vdash M{:}\,\sigma$. These can be interpreted as conditional statements saying that, for any environment ξ, if $\xi(x) \in [\![\tau]\!]^{\mathcal{D}}$ whenever $x{:}\,\tau \in \Gamma$, then the denotation of M under ξ is in the interpretation of σ. If $\models_{\mathcal{D},\xi} \Gamma$ abbreviates $\models_{\mathcal{D},\xi} x{:}\,\tau$ for all $x{:}\,\tau \in \Gamma$, then we define the semantical counterpart of $\Gamma \vdash M{:}\,\sigma$ as follows:

$$\Gamma \models_{\mathcal{D}} M{:}\,\sigma \Leftrightarrow (\forall \xi \in Var \to D)[\models_{\mathcal{D},\xi} \Gamma \Rightarrow \models_{\mathcal{D},\xi} M{:}\,\sigma].$$

The minimal requirement for the definition of type interpretation is the *soundness* of the type assignment system w.r.t. the given interpretation, namely that:

$$\text{if } \Gamma \vdash M{:}\,\sigma \text{ is derivable then } \Gamma \models_{\mathcal{D}} M{:}\,\sigma.$$

As the "meaning" of types is, up to now, given by the rules of the assignment system, we will pattern the definition of $[\![\sigma]\!]^{\mathcal{D}} \subseteq D$ after the rules themselves [17]. To simplify matters, we only consider the system BCD.

The easiest case is that of ω. As any term has type ω, this is interpreted as the property that is true of any object in the domain:

$$\Gamma \vdash M{:}\,\omega \quad (\omega) \quad \text{gives} \quad [\![\omega]\!]^{\mathcal{D}} = D.$$

The interpretation of $\sigma \wedge \tau$ is also clear: if something has both the property σ and the property τ, then it has the property $\sigma \wedge \tau$:

$$\frac{\Gamma \vdash M{:}\,\sigma \quad \Gamma \vdash M{:}\,\tau}{\Gamma \vdash M{:}\,\sigma \wedge \tau}\,(\wedge\mathrm{I}) \quad \text{which gives} \quad [\![\sigma \wedge \tau]\!]^{\mathcal{D}} \supseteq [\![\sigma]\!]^{\mathcal{D}} \cap [\![\tau]\!]^{\mathcal{D}}.$$

Using the ($\wedge$E) rules (which are derived rules, as remarked at page 64) we also have that $[\![\sigma \wedge \tau]\!]^{\mathcal{D}} \subseteq [\![\sigma]\!]^{\mathcal{D}} \cap [\![\tau]\!]^{\mathcal{D}}$, hence we get the equality: $[\![\sigma \wedge \tau]\!]^{\mathcal{D}} = [\![\sigma]\!]^{\mathcal{D}} \cap [\![\tau]\!]^{\mathcal{D}}$.

The interesting case is that of arrow types. Let us first consider the arrow introduction rule, namely:

$$\frac{\Gamma, x{:}\,\sigma \vdash M{:}\,\tau}{\Gamma \vdash \lambda x.M{:}\,\sigma \to \tau}\,(\to\mathrm{I})$$

The semantical reading of the premise is (forgetting about Γ):

$$(\forall \xi)[\xi(x) \in [\![\sigma]\!]^{\mathcal{D}} \Rightarrow [\![M]\!]^{\mathcal{D}}_{\xi} \in [\![\tau]\!]^{\mathcal{D}}].$$

Therefore for any $d \in [\![\sigma]\!]^{\mathcal{D}}$ it is the case that $[\![M]\!]^{\mathcal{D}}_{\xi[x:=d]} \in [\![\tau]\!]^{\mathcal{D}}$. In any λ-model this is the same as $[\![\lambda x.M]\!]^{\mathcal{D}}_{\xi} \cdot d \in [\![\tau]\!]^{\mathcal{D}}$.

If $F_D = \{e \in D \mid (\exists x, M, \xi)[e = [\![\lambda x.M]\!]^{\mathcal{D}}_{\xi}\}$, then rule ($\to$I) says:

$$\{e \in F_D \mid (\forall d)[d \in [\![\sigma]\!]^{\mathcal{D}} \Rightarrow e \cdot d \in [\![\tau]\!]^{\mathcal{D}}]\} \subseteq [\![\sigma \to \tau]\!]^{\mathcal{D}}.$$

On the other hand, the semantical reading of rule

$$\frac{\Gamma \vdash M{:}\,\sigma \to \tau \quad \Gamma \vdash N{:}\,\sigma}{\Gamma \vdash MN{:}\,\tau}\,(\to\mathrm{E})$$

is

$$[\![M]\!]^{\mathcal{D}}_{\xi} \in [\![\sigma \to \tau]\!]^{\mathcal{D}} \text{ and } [\![N]\!]^{\mathcal{D}}_{\xi} \in [\![\sigma]\!]^{\mathcal{D}} \Rightarrow [\![MN]\!]^{\mathcal{D}}_{\xi} \in [\![\tau]\!]^{\mathcal{D}},$$

that is

$$[\![\sigma \to \tau]\!]^{\mathcal{D}} \subseteq \{e \in D \mid (\forall d)[d \in [\![\sigma]\!]^{\mathcal{D}} \Rightarrow e \cdot d \in [\![\tau]\!]^{\mathcal{D}}]\}.$$

To settle the question of soundness, subsumption remains to be considered:

$$\frac{\Gamma \vdash M{:}\,\sigma \quad \sigma \leq_{\eta} \tau}{\Gamma \vdash M{:}\,\tau}\,(\mathrm{sub})$$

Whatever the interpretation of the premise $\sigma \leq_{\eta} \tau$ could be, this rule says that:

$$[\![M]\!]^{\mathcal{D}}_{\xi} \in [\![\sigma]\!]^{\mathcal{D}} \text{ and } \sigma \leq_{\eta} \tau \Rightarrow [\![M]\!]^{\mathcal{D}}_{\xi} \in [\![\tau]\!]^{\mathcal{D}}, \tag{3}$$

hence if $\sigma \leq_\eta \tau$ we expect $[\![\sigma]\!]^{\mathcal{D}} \subseteq [\![\tau]\!]^{\mathcal{D}}$. In particular, since $\omega \leq_\eta \omega \to \omega$, we have

$$[\![M]\!]^{\mathcal{D}}_{\xi} \in [\![\omega]\!]^{\mathcal{D}} \Rightarrow [\![M]\!]^{\mathcal{D}}_{\xi} \in [\![\omega \to \omega]\!]^{\mathcal{D}},$$

for all $M, \mathcal{D}, \xi$, i.e. by above $[\![\omega \to \omega]\!]^{\mathcal{D}} = D$. To get this we assume

$$[\![\sigma \to \tau]\!]^{\mathcal{D}} = \{e \in D \mid (\forall d)[d \in [\![\sigma]\!]^{\mathcal{D}} \Rightarrow e \cdot d \in [\![\tau]\!]^{\mathcal{D}}]\}.$$

We recast all this into a definition, which also takes into account the need of fixing an interpretation of type variables.

Definition 10.1 *Suppose that $\mathcal{D}$ is a λ-model. A* type environment *is a map $\zeta : \mathcal{TV} \to \mathcal{P}(D)$. Then we define $[\![\sigma]\!]^{\mathcal{D}}_{\zeta}$:*

1. $[\![\varphi]\!]^{\mathcal{D}}_{\zeta} = \zeta(\varphi)$, *for all* $\varphi \in \mathcal{TV}$,
2. $[\![\omega]\!]^{\mathcal{D}}_{\zeta} = D$,
3. $[\![\sigma \wedge \tau]\!]^{\mathcal{D}}_{\zeta} = [\![\sigma]\!]^{\mathcal{D}}_{\zeta} \cap [\![\tau]\!]^{\mathcal{D}}_{\zeta}$,
4. $[\![\sigma \to \tau]\!]^{\mathcal{D}}_{\zeta} = \{e \in D \mid (\forall d)[d \in [\![\sigma]\!]^{\mathcal{D}}_{\zeta} \Rightarrow e \cdot d \in [\![\tau]\!]^{\mathcal{D}}_{\zeta}]\}$.

We leave to the reader to verify that, if $\sigma \leq_\eta \tau$, then for any λ-model $\mathcal{D}$ and type environment ζ, $[\![\sigma]\!]^{\mathcal{D}}_{\zeta} \subseteq [\![\tau]\!]^{\mathcal{D}}_{\zeta}$.

We remark that, after the introduction of type environments in the definition of type interpretation, we have to amend our definition of the meaning of judgments, by writing

$$\models_{\mathcal{D},\xi,\zeta} M : \sigma \Leftrightarrow [\![M]\!]^{\mathcal{D}}_{\xi} \in [\![\sigma]\!]^{\mathcal{D}}_{\zeta}.$$

The meaning of $\Gamma \models_{\mathcal{D},\zeta} M : \sigma$ is defined accordingly.

The semantic counterparts of the typing statements in the assignment systems are naturally given by:

$$\Gamma \models M : \sigma \Leftrightarrow (\forall \mathcal{D}, \zeta)[\Gamma \models_{\mathcal{D},\zeta} M : \sigma].$$

Before stating the soundness theorem, we summarize in a definition the semantic notions we have introduced.

Definition 10.2 *Let $\mathcal{D}$ be a λ-model, ξ and ζ a term environment and a type environment over $\mathcal{D}$, respectively.*

1. $\models_{\mathcal{D},\xi,\zeta} M : \sigma \Leftrightarrow [\![M]\!]^{\mathcal{D}}_{\xi} \in [\![\sigma]\!]^{\mathcal{D}}_{\zeta}$
2. $\models_{\mathcal{D},\xi,\zeta} \Gamma \Leftrightarrow (\forall x : \tau \in \Gamma)[\models_{\mathcal{D},\xi,\zeta} x : \tau] \Leftrightarrow (\forall x : \tau \in \Gamma)[\xi(x) \in [\![\tau]\!]^{\mathcal{D}}_{\zeta}]$
3. $\Gamma \models_{\mathcal{D},\zeta} M : \sigma \Leftrightarrow (\forall \xi)[\models_{\mathcal{D},\xi,\zeta} \Gamma \Rightarrow \models_{\mathcal{D},\xi,\zeta} M : \sigma]$
4. $\Gamma \models M : \sigma \Leftrightarrow (\forall \mathcal{D}, \zeta)[\Gamma \models_{\mathcal{D},\zeta} M : \sigma]$.

The following theorem is proved by an easy induction on derivations [17]:

Theorem 10.3 (Soundness) $\Gamma \vdash_\eta M : \sigma \Rightarrow \Gamma \models M : \sigma$.

11 Filter Models and Completeness

Let us describe more abstractly and more generally what has been done in the previous section. A model of intersection types is a structure

$$\mathcal{A} = \langle A, \sqsubseteq, \sqcap, \top, \hookrightarrow \rangle$$

such that $\langle A, \sqsubseteq, \sqcap \rangle$ is an inf semi-lattice, $\top$ is the top in A, $\hookrightarrow$ a binary operation over A, and $\sqsubseteq$ satisfies the following axioms and rules:

$$\begin{array}{lcll} (a \hookrightarrow b) \sqcap (a \hookrightarrow c) & \sqsubseteq & a \hookrightarrow b \sqcap c & (\hookrightarrow \sqcap) \\ a' \sqsubseteq a, b \sqsubseteq b' & \Rightarrow & a \hookrightarrow b \sqsubseteq a' \hookrightarrow b' & (\hookrightarrow \sqsubseteq) \\ \top & \sqsubseteq & \top \hookrightarrow \top & (\top\text{-}\eta). \end{array}$$

In the literature such a structure is called an *extended applicative type structure*, EATS [29, 75]. We will essentially follow these two references in the whole development of the present section and of the following one. Previous section gives an example of an extended applicative type structure in which A is the powerset of D, $\sqsubseteq$ is subset inclusion, $\sqcap$ is set intersection, $\top$ is D and $\hookrightarrow$ the function space constructor.

We also have a satisfaction relation $\models \subseteq D \times A$, to model typing judgments, which gives two mappings:

$$\begin{array}{lll} ext : A \to \mathcal{P}(D) & \text{where} & a \mapsto \{d \mid d \models a\}, \\ prop : D \to \mathcal{P}(A) & \text{where} & d \mapsto \{a \mid d \models a\}. \end{array}$$

In the previous section $d \models a$ is $d \in a$, and *ext* is the identity. More generally we can think of $d \models a$ as stating "the element d satisfies the property a".

The order on A is such that

$$a \sqsubseteq b \Rightarrow ext(a) \subseteq ext(b),$$

which is to say

$$d \models a, a \sqsubseteq b \Rightarrow d \models b.$$

As A is an inf semi-lattice, for any $a, b \in A$, $ext(a \sqcap b) \subseteq ext(a) \cap ext(b)$. On the other hand we have:

$$d \models a, d \models b \Rightarrow d \models a \sqcap b,$$

which implies $ext(a \sqcap b) = ext(a) \cap ext(b)$. We also observe that there is a top element $\top \in A$ such that $d \models \top$ for any d: it comes out that

$$\top \in prop(d) \neq \emptyset.$$

Under these conditions, for any $d \in D$ the set $prop(d)$ is a *filter*, namely a subset $F \subseteq A$ such that:

$$a \in F, a \sqsubseteq b \quad \Rightarrow \quad b \in F, \tag{4}$$

$$a, b \in F \;\Rightarrow\; a \sqcap b \in F,$$

$$\top \in F.$$

The kernel of *prop* induces over D an equivalence relation $\sim$:

$$d \sim e \Leftrightarrow prop(d) = prop(e).$$

Taking the quotient $D/_{\sim}$ amounts to identifying objects having the same properties. Equivalence classes are injectively mapped, by construction, into the set $\mathcal{F}(A)$ of filters over A. Vice versa, we can think of $\mathcal{F}(A)$ as the set of "abstract objects", which includes, but does not necessarily coincides with, the image of $D/_{\sim}$.

How can we talk of properties of "abstract objects"? One way to do that is to look for a map $\vartheta : A \to \mathcal{P}(\mathcal{F}(A))$ that reinterprets properties in A consistently with our view of $\mathcal{F}(A)$ as the set of abstract objects, i.e., a map ϑ such that:

$$d \models a \Rightarrow prop(d) \in \vartheta(a).$$

Now if $d \models a$ then $a \in prop(d)$, i.e., $prop(d) \in \{F \in \mathcal{F}(A) \mid a \in F\}$. Therefore a natural choice is

$$\vartheta(a) = \{F \in \mathcal{F}(A) \mid a \in F\}.$$

This map is monotonic w.r.t. to subset inclusion, namely

$$a \sqsubseteq b \Rightarrow \vartheta(a) \subseteq \vartheta(b),$$

as if $F \in \vartheta(a)$ then $a \in F$, so that $b \in F$ by (4), hence $F \in \vartheta(b)$. Moreover ϑ preserves $\sqcap$ in the sense that

$$\vartheta(a \sqcap b) = \vartheta(a) \cap \vartheta(b),$$

and it is injective, since

$$a \not\sqsubseteq b \Rightarrow b \notin\uparrow a \Rightarrow\uparrow a \in \vartheta(a) \text{ and } \uparrow a \notin \vartheta(b) \Rightarrow \vartheta(a) \neq \vartheta(b),$$

where $\uparrow a = \{c \in A \mid a \sqsubseteq c\}$ is the *principal filter* generated by a. In conclusion:

Proposition 11.1 *If $\langle A, \sqsubseteq, \sqcap\rangle$ is an inf semi-lattice, then it is isomorphic to a subalgebra of the inf semi-lattice $\langle \mathcal{P}(\mathcal{F}(A)), \subseteq, \cap\rangle$ via the map ϑ.*

We can now rephrase the definition of interpretation of the arrow as follows:

$$d \models a \hookrightarrow b \Leftrightarrow (\forall e \in D)[e \models a \Rightarrow d \cdot e \models b]. \tag{5}$$

We have an implicit notion of neighborhoods over D, namely a set $ext(a)$ for each $a \in A$. Then the familiar definition of continuity becomes:

$$f(y) \models b \Rightarrow (\exists a \in A)[(y \models a) \text{ and } (\forall z)(z \models a \Rightarrow f(z) \models b)]. \tag{6}$$

In the present construction, for any $d \in D$ the function $\Phi(d)$ is continuous, therefore, by (6):

$$d \cdot e \models b \Rightarrow (\exists a \in A)[e \models a \text{ and } (\forall e')(e' \models a \Rightarrow d \cdot e' \models b)].$$

Because of (5) and $(\top - \eta)$, this is equivalent to:

$$d \cdot e \models b \Leftrightarrow (\exists a \in A)[e \models a \text{ and } d \models a \hookrightarrow b]$$

i.e., to:

$$prop(d \cdot e) = \{b \mid (\exists a \in A)[a \in prop(e) \text{ and } (a \hookrightarrow b) \in prop(d)]\}.$$

We have thus defined an operation on the image of *prop*:

$$prop(d) \cdot prop(e) = prop(d \cdot e).$$

If we extend this operation to arbitrary "abstract objects", namely to the filters in $\mathcal{F}(A)$, we obtain

$$F \cdot F' = \{b \mid (\exists a \in A)[a \in F' \text{ and } (a \hookrightarrow b) \in F\}.$$

As a matter of fact, using $(\hookrightarrow \sqcap)$, $(\hookrightarrow \sqsubseteq)$, and $(\top\text{-}\eta)$, we can show that $\mathcal{F}(A)$ is closed under application.

Proposition 11.2 *If A is an EATS, and $F, F' \in \mathcal{F}(A)$, then $F \cdot F' \in \mathcal{F}(A)$. Moreover*

$$\vartheta(a \hookrightarrow b) = \{F \mid (\forall F')[F' \in \vartheta(a) \Rightarrow F \cdot F' \in \vartheta(b)]\},$$

so that, if we define

$$U \rightsquigarrow V = \{F \mid (\forall F')[F' \in U \Rightarrow F \cdot F' \in V]\},$$

for any $U, V \in \mathcal{P}(\mathcal{F}(A))$, then $\langle \mathcal{P}(\mathcal{F}(A)), \subseteq, \cap, \mathcal{F}(A), \rightsquigarrow \rangle$ is an EATS, and ϑ is an EATS isomorphism of A into a subalgebra of $\mathcal{P}(\mathcal{F}(A))$.

Because of this proposition, $\mathcal{F}(A)$ is now an applicative structure. We may define a mapping $\Phi^{\mathcal{F}}$ whose domain is $\mathcal{F}(A)$, such that

$$\Phi^{\mathcal{F}}(F) = \lambda X.F \cdot X \in \mathcal{F}(A)^{\mathcal{F}(A)}.$$

What can we say about the range of $\Phi^{\mathcal{F}}$? Let $h : \mathcal{F}(A) \rightarrow \mathcal{F}(A)$ be a function; adapting (6), we say that it is continuous if for all F and $b \in A$:

$$h(F) \in \vartheta(b) \Rightarrow (\exists a \in A)[F \in \vartheta(a) \text{ and } h[\vartheta(a)] \subseteq \vartheta(b)]. \tag{7}$$

If h is continuous and $b \in h(F)$, then $b \in h(\uparrow a)$ for some $a \in F$. Vice versa if $b \in h(\uparrow a)$ for some $a \in F$, then there exists $c \in \uparrow a \subseteq F$ such that $h[\vartheta(c)] \subseteq \vartheta(b)$.

But $F \in \vartheta(c)$, therefore $h(F) \in \vartheta(b)$, i.e. $b \in h(F)$. We conclude that if h is continuous, then for all F

$$h(F) = \bigcup_{a \in F} h(\uparrow a). \tag{8}$$

That (8) implies (7) is also true, so that these are equivalent definitions of continuity. The set of continuous functions over $\mathcal{F}(A)$ will be denoted by $[\mathcal{F}(A) \to \mathcal{F}(A)]$. Now, through routine calculations, we establish

$$F \cdot F' = \bigcup_{a \in F, b \in F'} (\uparrow a \cdot \uparrow b).$$

This says that application is continuous in both its arguments: continuity in the second argument implies $Rng(\Phi^{\mathcal{F}}) \subseteq [\mathcal{F}(A) \to \mathcal{F}(A)]$. Continuity in the first argument implies

$$\Phi^{\mathcal{F}}(F) = \bigcup_{a \in F} \Phi^{\mathcal{F}}(\uparrow a),$$

that is $\Phi^{\mathcal{F}}$ itself is "continuous" (at least it satisfies a similar condition as (8)).

But we can show more: under a suitable hypothesis $[\mathcal{F}(A) \to \mathcal{F}(A)]$ is the range of $\Phi^{\mathcal{F}}$. Aiming at this, we first observe that a continuous function h is completely determined by its behavior on principal filters, namely by pairs $a, b \in A$ such that $\uparrow b \subseteq h(\uparrow a)$, as this immediately follows from (8), since $F = \bigcup_{a \in F} \uparrow a$. On the other hand, the definition of application suggests a possible coding of a continuous function h into a filter: take the least filter F such that, if $\uparrow b \subseteq h(\uparrow a)$ then $a \hookrightarrow b \in F$. This defines a function $\Psi^{\mathcal{F}} : [\mathcal{F}(A) \to \mathcal{F}(A)] \to \mathcal{F}(A)$ by

$$\Psi^{\mathcal{F}}(h) = \bigcap \{F \in \mathcal{F}(A) \mid \uparrow b \subseteq h(\uparrow a) \Rightarrow a \hookrightarrow b \in F\}.$$

Remark that, if $[\mathcal{F}(A) \to \mathcal{F}(A)]$ is taken with the pointwise ordering, then $\Psi^{\mathcal{F}}$ is "continuous" in the sense that it preserves directed sups.

We say that an EATS is *continuous* if and only if it satisfies the following condition:

$$\sqcap_{i \in I}(a_i \hookrightarrow b_i) \sqsubseteq a \hookrightarrow b \Rightarrow \sqcap\{b_i \mid i \in I, a \sqsubseteq a_i\} \sqsubseteq b \qquad \text{(cont)}$$

where I is a finite set of indices. As a matter of fact, because of $(\hookrightarrow \sqcap)$, in any EATS the opposite implication of (cont) holds, so that in a continuous EATS (cont) is actually an equivalence. Notice that $\leq_\eta$ satisfies (cont), as stated in Proposition 9.6.

Theorem 11.3 *Let $\mathcal{A}$ be a continuous EATS. Then $\langle \mathcal{F}(A), \Phi^{\mathcal{F}}, \Psi^{\mathcal{F}} \rangle$ is a λ-model (filter model), that is*

$$\Phi^{\mathcal{F}} \circ \Psi^{\mathcal{F}} = Id.$$

Moreover, if the term interpretation map is defined via $\Phi^{\mathcal{F}}$ and $\Psi^{\mathcal{F}}$, then for any term M and term environment $\xi : Var \to \mathcal{F}(A)$:

$$[\![M]\!]^{\mathcal{F}(A)}_{\xi[x:=F]} = \bigcup_{a \in F} [\![M]\!]^{\mathcal{F}(A)}_{\xi[x:=\uparrow a]}.$$

Proof. We defer the proof of $\Phi^{\mathcal{F}} \circ \Psi^{\mathcal{F}} = Id$ to the next section (Corollary 12.4). The second part of the theorem follows by an induction over M: it says that any function $\lambda F. [\![M]\!]^{\mathcal{F}(A)}_{\xi[x:=F]}$ is continuous, therefore it is in the domain of $\Psi^{\mathcal{F}}$. □

The construction of a filter model is interesting since it is a tool for proving that the intersection type assignment system BCD is complete w.r.t. the semantics introduced in Section 10.

Consider the EATS $\langle \mathcal{T}, \leq_\eta, \wedge, \omega, \to \rangle$, where $\leq_\eta$ is the BCD subtype relation, and where $\mathcal{T}$ is the (quotient under $\leq_\eta$ of) the set of types: then this EATS is continuous by Proposition 9.6. Let $\zeta_0 : \mathcal{TV} \to \mathcal{P}(\mathcal{F}(\mathcal{T}))$ be the type environment defined by:

$$\zeta_0(\varphi) = \vartheta(\varphi) = \{F \in \mathcal{F}(\mathcal{T}) \mid \varphi \in F\}.$$

It is not difficult to prove that, for all $\sigma \in \mathcal{T}$,

$$[\![\sigma]\!]^{\mathcal{F}(\mathcal{T})}_{\zeta_0} = \vartheta(\sigma) = \{F \in \mathcal{F}(\mathcal{T}) \mid \sigma \in F\}.$$

But ϑ is an isomorphism, therefore

$$\sigma \leq_\eta \tau \Rightarrow \vartheta(\sigma) \subseteq \vartheta(\tau).$$

Now the key step consists in showing that the interpretation of any term M in the λ-model $\mathcal{F}(\mathcal{T})$ coincides with the set of types derivable for M in the system BCD:

Theorem 11.4 *The interpretation of terms into the λ-model $\mathcal{F}(\mathcal{T})$ is such that, for any term M and term environment ξ:*

$$[\![M]\!]^{\mathcal{F}(\mathcal{T})}_{\xi} = \{\sigma \mid (\exists \Gamma)[\models_{\mathcal{F}(\mathcal{T}),\xi,\zeta_0} \Gamma \text{ and } \Gamma \vdash_\eta M{:}\sigma]\}.$$

To help the reader, we first observe that $\models_{\mathcal{F}(\mathcal{T}),\xi,\zeta_0} \Gamma$ if and only if for all $x{:}\tau \in \Gamma$, we get $\xi(x) \in [\![\tau]\!]^{\mathcal{F}(\mathcal{T})}_{\zeta_0} = \vartheta(\tau)$, namely that $\tau \in \xi(x)$ (which makes sense, as $\xi(x) \in \mathcal{F}(\mathcal{T})$).

We only sketch the proof of Theorem 11.4. It is by cases, going through the definition of $[\![M]\!]^{\mathcal{F}(\mathcal{T})}_{\xi}$. In the case of the application one establishes the thesis by using Lemma 9.5(4)

$$\Gamma \vdash_\eta MN{:}\tau \Leftrightarrow (\exists \sigma)[\Gamma \vdash_\eta M{:}\sigma \to \tau \text{ and } \Gamma \vdash_\eta N{:}\sigma].$$

Similarly, in the case of abstraction the thesis follows by

$$\Gamma \vdash_\eta \lambda x.M{:}\sigma \to \tau \Leftrightarrow \Gamma, x{:}\sigma \vdash_\eta M{:}\tau$$

which is Lemma 9.5(3). We have now all we need for the proof of the completeness theorem (first given in [17]).

Theorem 11.5 (Completeness) $\Gamma \vdash_\eta M{:}\sigma \Leftrightarrow \Gamma \models M{:}\sigma$.

Proof. The only if part is Theorem 10.3. For the if part, it suffices that for some $\lambda$-model $\mathcal{D}$ and type environment ζ, $\Gamma \models_{\mathcal{D},\zeta} M{:}\,\sigma$ implies $\Gamma \vdash_\eta M{:}\,\sigma$.

Given a basis $\Gamma$ set:

$$\xi_\Gamma(x) = \begin{cases} \uparrow\sigma & \text{if } x{:}\,\sigma \in \Gamma \\ \uparrow\omega & \text{otherwise.} \end{cases}$$

By construction $\models_{\mathcal{F}(\mathcal{T}),\xi_\Gamma,\zeta_0} \Gamma$. Now

$$\begin{array}{rcll} \Gamma \models_{\mathcal{F}(\mathcal{T}),\zeta_0} M{:}\,\sigma & \Rightarrow & [\![M]\!]^{\mathcal{F}(\mathcal{T})}_{\xi_\Gamma} \in [\![\sigma]\!]^{\mathcal{F}(\mathcal{T})}_{\zeta_0} = \vartheta(\sigma) & \\ & \Rightarrow & \sigma \in [\![M]\!]^{\mathcal{F}(\mathcal{T})}_{\xi_\Gamma} & \\ & \Rightarrow & (\exists \Gamma')[\models_{\mathcal{F}(\mathcal{T}),\xi_\Gamma,\zeta_0} \Gamma' \text{ and } \Gamma' \vdash_\eta M{:}\,\sigma] & \text{by Theorem 11.4.} \end{array}$$

If $\models_{\mathcal{F}(\mathcal{T}),\xi_\Gamma,\zeta_0} \Gamma'$ then for any $x{:}\,\tau' \in \Gamma'$, $\tau' \in \xi_\Gamma(x)$; therefore there exists $\tau$ such that $x{:}\,\tau \in \Gamma$ and $\tau \leq_\eta \tau'$. It is easy to check that:

$$\Gamma' \Subset \Gamma \text{ and } \Gamma' \vdash_\eta M{:}\,\sigma \Rightarrow \Gamma \vdash_\eta M{:}\,\sigma.$$

Concluding we have that

$$\Gamma \models_{\mathcal{F}(\mathcal{T}),\zeta_0} M{:}\,\sigma \Rightarrow \Gamma \vdash_\eta M{:}\,\sigma,$$

and the thesis follows. □

12 Filter Models and Domains

Filter models arising from EATS turn out to be useful for describing a whole class of domains. Let $A$ be an inf semi-lattice and $\mathcal{F}(A)$ its filter structure. Then $\mathcal{F}(A)$ is naturally ordered by set inclusion. Moreover, if $\mathcal{G} \subseteq \mathcal{F}(A)$ then $\bigcap \mathcal{G}$ is a filter. Now any ordered set closed under arbitrary infs has also arbitrary sups:

$$\bigsqcup \mathcal{G} = \uparrow\{b \mid \exists n\, \exists a_1, \ldots, a_n \in \bigcup \mathcal{G}.\ b = a_1 \sqcap \cdots \sqcap a_n\},$$

where by $\uparrow X$ for $X \subseteq A$ we mean the upper closure of $X$.

We remark that, if $\mathcal{G}$ is directed with respect to $\subseteq$, then $\bigcup \mathcal{G}$ is a filter, so that in this case $\bigsqcup \mathcal{G}$ is just the union of $\mathcal{G}$. We conclude that $\langle \mathcal{F}(A), \subseteq \rangle$ is a complete lattice.

The inf semi-lattice $A$ is injectively embedded into $\mathcal{F}(A)$ via the map $\uparrow\_ : A \to \mathcal{F}(A)$, which is however order reversing since

$$a \sqsubseteq b \Leftrightarrow \uparrow b \subseteq \uparrow a.$$

On the other hand the image of $A$ into $\mathcal{F}(A)$ determines its structure, as we have shown that $F = \bigcup_{a \in F} \uparrow a$ for any $F \in \mathcal{F}(A)$. It is worth to note that $\{\uparrow a \mid a \in F\}$ is a directed subset of $\mathcal{F}(A)$, so that we can read the last equation as the algebraicity property of $\mathcal{F}(A)$[5].

[5]See the classic [50], Ch. 1, §4. A more recent reference is [88], Ch. 3.

Proposition 12.1 *If $A$ is an inf semi-lattice, then $\langle \mathcal{F}(A), \subseteq \rangle$ is a complete algebraic lattice, whose finite elements*[6] *$\mathcal{KF}(A)$ are exactly the principal filters, namely the image of $A$ via $\uparrow\_$.*

Proof. If $\mathcal{G}$ is a directed subset of $\mathcal{F}(A)$ and $\uparrow a \subseteq \bigsqcup \mathcal{G}$ then $\bigsqcup \mathcal{G} = \bigcup \mathcal{G}$ so that $a \in \uparrow a \subseteq G$ for some $G \in \mathcal{G}$: hence $\uparrow a \in \mathcal{KF}(A)$. Vice versa if $F \in \mathcal{KF}(A)$ then, from $F = \bigcup_{a \in F} \uparrow a$, we immediately conclude that $F = \uparrow a$ for some $a \in F$. □

The last proposition is the half of a representation theorem for complete algebraic lattices. Indeed, if $D$ is such a structure and $d, e \in \mathcal{K}(D)$, then $d \sqcup e \in \mathcal{K}(D)$, so that $\mathcal{K}(D)$ is a sup semi-lattice, and therefore $\mathcal{K}(D)^{op}$ is an inf semi-lattice[7]. We know from domain theory that any algebraic dcpo (hence in particular any complete algebraic lattice) is isomorphic to the ideal completion $\mathcal{I}(\mathcal{K}(D))$ of its finite elements. Now ideals are dual to filters, so that if we reverse the order, ideals become filters and

$$\mathcal{F}(\mathcal{K}(D)^{op}) = \mathcal{I}(\mathcal{K}(D)) \simeq D.$$

Corollary 12.2 (Representation of Complete Algebraic Lattices) *A poset $\langle D, \sqsubseteq \rangle$ is a complete algebraic lattice if and only if there exists some inf semi-lattice $A$ such that $D \simeq \mathcal{F}(A)$, or, dually, if and only if $D \simeq \mathcal{I}(A^{op})$.*

Domains are seen both as ordered structures and as topological spaces. As such they are taken with the Scott topology. One way to present this topology is to consider the subbasis consisting of the opens

$$\mathcal{O}_e = \{d \in D \mid e \sqsubseteq d\}, \qquad \text{where} \qquad e \in \mathcal{K}(D).$$

Now in the case of $D = \mathcal{F}(A)$ we have

$$\mathcal{O}_{\uparrow a} = \{F \in \mathcal{F}(A) \mid \uparrow a \subseteq F\} = \{F \in \mathcal{F}(A) \mid a \in F\} = \vartheta(a).$$

Therefore Proposition 11.1 can be strengthened by saying that the subalgebra of $\mathcal{P}(\mathcal{F}(A))$ to which $\langle A, \sqsubseteq, \sqcap \rangle$ is isomorphic via ϑ is a subbasis of the Scott topology $\Omega\mathcal{F}(A)$ over $\mathcal{F}(A)$.

This construction is universal. Define a *frame* as a partial order closed under finite infs and arbitrary sups, which satisfies the frame distribution law:

$$\bigsqcup Y \sqcap x = \bigsqcup \{y \sqcap x \mid y \in Y\}.$$

A special case of frame is the set of opens of a topological space. Frames form a category, whose morphisms are mappings preserving $\bigsqcup$ and $\sqcap$. Let $A$ be an inf semi-lattice and $B$ be a frame: observe that a frame is also an inf semi-lattice.

[6]In a cpo $D$ an element $d$ is finite, or compact, if whenever $X \subseteq D$ is directed and $d \sqsubseteq \bigsqcup X$ then $d \sqsubseteq e$ for some $e \in X$. The set of finite elements of $D$ will be denoted by $\mathcal{K}(D)$.

[7]What makes the difference between generic algebraic dcpos and algebraic lattices is that in the former case $\mathcal{K}(D)$ is not necessarily a sup semi-lattice, but just a conditional sup semi-lattice, which has the sups of upper bounded elements. See e.g. [88], §3.2.

Then for any inf semi-lattice morphism $f : A \to B$ there exists a unique frame morphism $f' : \Omega\mathcal{F}(A) \to B$ such that $f = f' \circ \vartheta$, whose definition is

$$f'(\mathcal{O}) = \bigsqcup\{f(a) \mid \vartheta(a) \subseteq \mathcal{O}\}.$$

In Section 11, Theorem 11.3, we deferred to the present section the proof that the range of $\Phi^{\mathcal{F}}$ is the space of continuous functions over $\mathcal{F}(A)$. Let us address this question now.

If $D$ is a complete algebraic lattice then the space of Scott continuous functions $[D \to D]$ is also a complete algebraic lattice[8]. The finite elements of $[D \to D]$ are then finite sups of *step functions*: if $d, e \in \mathcal{K}(D)$ then the step function $(d \Rightarrow e)$ is defined by:

$$(d \Rightarrow e)(x) = \begin{cases} e & \text{if } d \sqsubseteq x \\ \bot & \text{otherwise.} \end{cases}$$

If we look at $D$ as a filter structure $\mathcal{F}(A)$, where $A$ is an EATS, then we see a relation between the principal filter $\uparrow(a \hookrightarrow b)$ (which is a finite element of $\mathcal{F}(A)$) and the step function $(\uparrow a \Rightarrow \uparrow b)$ (which is finite in $[\mathcal{F}(A) \to \mathcal{F}(A)]$) given by the map $\Phi^{\mathcal{F}}$, or, equivalently, by the operation $\_\cdot\_$:

$$\uparrow(a \hookrightarrow b) \cdot G = \begin{cases} \uparrow b & \text{if } \uparrow a \subseteq G \\ \uparrow\top & \text{else,} \end{cases}$$

for any filter $G$; therefore $\Phi^{\mathcal{F}}(\uparrow(a \hookrightarrow b)) = (\uparrow a \Rightarrow \uparrow b)$, since $\uparrow\top$ is the bottom of $\mathcal{F}(A)$. The same thing can be observed on the topological side. If $a \hookrightarrow b \in F$ then $\uparrow b \subseteq F \cdot \uparrow a$. But also the vice versa is true: indeed if $\uparrow b \subseteq F \cdot \uparrow a$ then for some $a' \in \uparrow a$ it is the case that $a' \hookrightarrow b \in F$. We conclude $a' \hookrightarrow b \sqsubseteq a \hookrightarrow b$, that is $a \hookrightarrow b \in F$. Therefore

$$a \hookrightarrow b \in F \Leftrightarrow b \in \Phi^{\mathcal{F}}(F)(\uparrow a) \Leftrightarrow (\uparrow a \Rightarrow \uparrow b) \sqsubseteq \Phi^{\mathcal{F}}(F), \tag{9}$$

namely $\Phi^{\mathcal{F}}$ sends the open set $\vartheta(a \hookrightarrow b)$ into the open set $\mathcal{O}_{(\uparrow a \Rightarrow \uparrow b)}$.

Since we know that $\uparrow((a \hookrightarrow b) \sqcap (a' \hookrightarrow b')) = \uparrow(a \hookrightarrow b) \sqcup \uparrow(a' \hookrightarrow b')$, it is tempting to conclude that we can describe via $\Phi^{\mathcal{F}}$ the whole set $\mathcal{K}[\mathcal{F}(A) \to \mathcal{F}(A)]$, and therefore that $[\mathcal{F}(A) \to \mathcal{F}(A)]$ is the range of $\Phi^{\mathcal{F}}$. This is, however, not the case in general, since even if we can show that $\Phi^{\mathcal{F}}$ is continuous, we cannot say anything about its behavior on (finite) undirected sups. Indeed the fact that the range of $\Phi^{\mathcal{F}}$ coincides with the space of continuous functions over $\mathcal{F}(A)$ actually depends on the axioms about the arrow holding in the EATS $\mathcal{A}$.

Theorem 12.3 *Let $\mathcal{A}$ be an EATS. Then the range of $\Phi^{\mathcal{F}}$ coincides with the set of Scott continuous functions over $\mathcal{F}(A)$ if and only if $\mathcal{A}$ is continuous.*

[8]It is known that, if $D$ is an algebraic lattice which is not complete, then $[D \to D]$ may fail to be algebraic. See e.g. [88], §3.3, Example 3.10.

Proof. We first observe that, if $f = \bigsqcup_{i \in I}(\uparrow a_i \Rightarrow \uparrow b_i)$ where I is a finite set of indexes, then for any $F \in \mathcal{F}(A)$ we have

$$\begin{aligned} f(F) &= \bigsqcup\{\uparrow b_i \mid i \in I, a_i \in F\} \\ &= \uparrow\sqcap\{b_i \mid i \in I, a_i \in F\}. \end{aligned} \tag{10}$$

Now, to prove the if part of the theorem we show that $f = \Phi^{\mathcal{F}}(\uparrow\sqcap_{i \in I}(a_i \hookrightarrow b_i))$ namely that all finite sups of step functions are representable: as $\Phi^{\mathcal{F}}$ is continuous, this implies that all continuous functions are in the range of $\Phi^{\mathcal{F}}$. We recall that, in a continuous EATS $\mathcal{A}$ it holds

$$\sqcap_{i \in I}(a_i \hookrightarrow b_i) \sqsubseteq a \hookrightarrow b \Leftrightarrow \sqcap\{b_i \mid i \in I, a \sqsubseteq a_i\} \sqsubseteq b. \tag{11}$$

Then we have that for any $c \in A$

$$\begin{aligned} & b \in \Phi^{\mathcal{F}}(\uparrow\sqcap_{i \in I}(a_i \hookrightarrow b_i))(\uparrow c) \\ & \Leftrightarrow (\exists a \in \uparrow c)[\sqcap_{i \in I}(a_i \hookrightarrow b_i) \sqsubseteq a \hookrightarrow b] \\ & \Leftrightarrow (\exists a \in \uparrow c)[\sqcap\{b_i \mid i \in I, a \sqsubseteq a_i\} \sqsubseteq b] \qquad \text{by (11)} \\ & \Leftrightarrow \sqcap\{b_i \mid i \in I, c \sqsubseteq a_i\} \sqsubseteq b \\ & \Leftrightarrow b \in f(\uparrow c). \end{aligned}$$

As f and $\Phi^{\mathcal{F}}(\uparrow\sqcap_{i \in I}(a_i \hookrightarrow b_i))$ agree on finite elements, by continuity they are the same.

For the only if part suppose that $\sqcap_{i \in I}(a_i \hookrightarrow b_i) \sqsubseteq a \hookrightarrow b$. Taking $f = \bigsqcup_{i \in I}(\uparrow a_i \Rightarrow \uparrow b_i)$ as above, there exists $G \in \mathcal{F}(A)$ s.t. $f = \Phi^{\mathcal{F}}(G)$, since $\Phi^{\mathcal{F}}$ is surjective. By (9), $a_i \hookrightarrow b_i \in G$ for all $i \in I$, therefore $\sqcap_{i \in I}(a_i \hookrightarrow b_i) \in G$ since G is closed under finite inf. But G is also upward closed, so that $a \hookrightarrow b \in G$ and $b \in \Phi^{\mathcal{F}}(G)(\uparrow a) = f(\uparrow a)$ again by (9). Using this fact and (10), we conclude that:

$$b \sqsupseteq \sqcap\{b_i \mid i \in I, a_i \in \uparrow a\} = \sqcap\{b_i \mid i \in I, a \sqsubseteq a_i\}.$$

□

Corollary 12.4 *If $\mathcal{F} = \mathcal{F}(A)$ is the space of filters over a continuous EATS $\mathcal{A}$, then $\Phi^{\mathcal{F}} \circ \Psi^{\mathcal{F}} = Id$.*

Proof. It suffices to check the identity $\Phi^{\mathcal{F}} \circ \Psi^{\mathcal{F}} = Id$ against compact functions. Take $f = \bigsqcup_{i \in I}(\uparrow a_i \Rightarrow \uparrow b_i)$: by similar calculations as in the proof of Theorem 12.3 we have

$$\begin{aligned} \Psi^{\mathcal{F}}(f) &= \bigcap\{F \mid b \in f(\uparrow a) \Rightarrow a \hookrightarrow b \in F\} \\ &= \bigcap\{F \mid \sqcap\{b_i \mid i \in I, a_i \in \uparrow a\} \sqsubseteq b \Rightarrow a \hookrightarrow b \in F\} \\ &= \bigcap\{F \mid \sqcap_{i \in I}(a_i \hookrightarrow b_i) \sqsubseteq a \hookrightarrow b \Rightarrow a \hookrightarrow b \in F\} \\ &= \uparrow\sqcap_{i \in I}(a_i \hookrightarrow b_i). \end{aligned}$$

From the proof of Theorem 12.3 we know that $\Phi^{\mathcal{F}}(\uparrow\sqcap_{i\in I}(a_i \hookrightarrow b_i)) = f$ and we are done. □

We refer to [29] for the proof that filter models can be used to mimic inverse limit constructions that solve domain equations.

We conclude with a few remarks concerning the importance of choosing certain EATS axioms for the arrow with respect to the semantics of arrow types and the basic retraction $[D \to D] \trianglelefteq D$[9]. We observed that putting $\omega \le \omega \to \omega$ in the type theory enforces

$$[\![\sigma \to \tau]\!]^{\mathcal{D}}_{\xi} = \{e \in D \mid (\forall d)[d \in [\![\sigma]\!]^{\mathcal{D}}_{\xi} \Rightarrow e \cdot d \in [\![\tau]\!]^{\mathcal{D}}_{\xi}]\}.$$

We may relax this, by taking $\sigma \to \omega \le \omega \to \omega$ instead, at the price of redefining the filter application operation by:

$$F \cdot F' = \{b \mid (\exists a \in A)[a \in F' \text{and } (a \hookrightarrow b) \in F]\} \cup \uparrow\omega.$$

To break the equality $\omega = \omega \to \omega$ is indeed necessary for modeling the lazy λ-calculus instead of the classical λ-calculus [2]. On the other hand if we want an extensional λ-model, whose domain equation is $D = [D \to D]$, then we ask for the equality $\Psi^{\mathcal{F}} \circ \Phi^{\mathcal{F}} = Id$. In these models, η-convertible terms must have the same meanings, i.e., the same types. This is true when any type σ is equivalent to a finite intersection of arrow types. Now, for a continuous EATS to induce an extensional λ-model it suffices that any type variable φ has this property [34].

13 Types and Böhm Trees

If we consider λ-terms as programs we are led naturally to the intuitive idea that the meaning of a λ-term (program) is represented by the amount of "meaningful information" we can extract from the term by "running it". The formalization of "the information" obtained from a term requires first the definition of what is, in a λ-term, a "stable relevant minimal information" directly observable in the term. This is the token of information which cannot be altered by further reductions but can only be added upon. (As an example the reader may think of the calculation of the $\sqrt{2}$. The calculation process merely adds decimals to the already calculated decimal expansion).
If one orders all the stable relevant minimal information which can be obtained from a computation according their information content, then a tree naturally arises. There are many tree representations in the literature, depending on the different notions of stable relevant minimal information. Among them Böhm trees [15] (Chapter 10) have been first introduced and widely studied. In this case we consider head normal forms as the only stable relevant information. We denote by Ω the absence of information. Therefore we are naturally led to the following definition of Böhm tree of a λ-term M ($BT(M)$).

[9]There is retraction of D into E, written $D \trianglelefteq E$, if there exist the morphisms $\psi : D \to E$ and $\varphi : E \to D$ s.t. $\varphi \circ \psi = Id$.

Definition 13.1 *The Böhm tree $BT(M)$ of a term M is defined by cases as follows:*

- *if M is solvable, i.e. $M \xrightarrow{*}_\beta \lambda x_1 \ldots x_n.yN_1 \ldots N_m$ $(n,m \geq 0)$, then:*

$$BT(M) = \begin{tikzcd} & \lambda x_1 \ldots x_n.y \arrow[dl, dash] \arrow[dr, dash] & \\ BT(N_1) & \ldots & BT(N_m) \end{tikzcd}$$

- *otherwise, i.e. if M is unsolvable:*

$$BT(M) = \Omega.$$

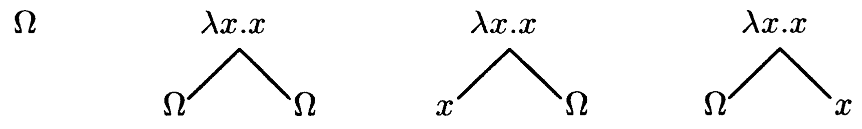
For example if $\Delta_3 \equiv \lambda x.xxx$ and $\Omega_3 \equiv \Delta_3\Delta_3$ we get:

$$BT(\Delta_3) = \begin{tikzcd} & \lambda x.x \arrow[dl, dash] \arrow[dr, dash] & \\ x & & x \end{tikzcd} \qquad BT(\Omega_3) = \Omega$$

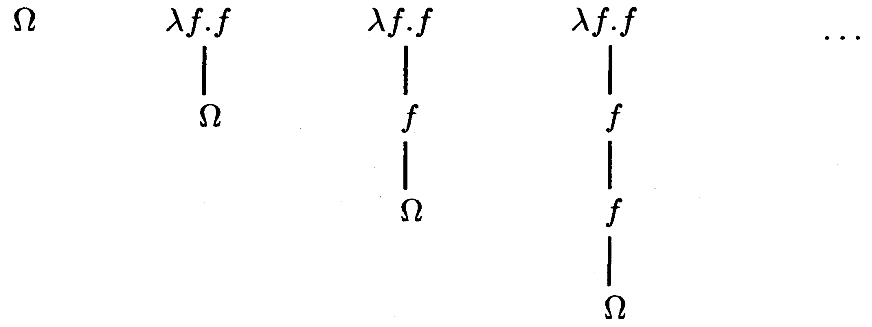
being Δ_3 a normal form and Ω_3 an unsolvable term.
The fixed-point combinator $\mathbf{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ reduces to $\lambda f.\, \overbrace{f(\ldots(f}^{n}(\mathbf{Y}f)\ldots)$ for every $n \geq 1$ and therefore we obtain the following infinite Böhm tree:

$$BT(Y) = \begin{tikzcd} \lambda f.f \arrow[d, dash] \\ f \arrow[d, dash] \\ f \\ \vdots \end{tikzcd}$$

The last example shows that we can end with infinite trees, which are effectively generated. The point is that, each time, we do not have computed the whole tree, but just a part of it, and something always remains to be computed. This can be represented by taking cuts of the infinite tree of finite depth, putting Ω as label of certain leaves to mean that there we lack of information. If finite trees match up to subtrees rooted in such undefined nodes, we say that they are compatible, and their merge is a sup of the information they convey. The infinite tree can now be recovered as the limit (sup) of its finite cuts. As a first example let us consider the cutting of the finite tree $BT(\Delta_3)$, which yields a finite set of trees:



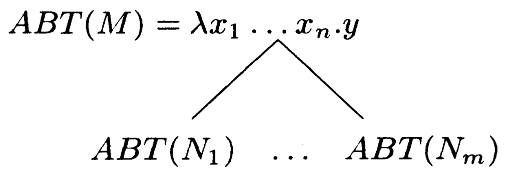
Then we apply the same technique to the infinite case, cutting $BT(Y)$ to an infinite set of trees:



It is important to notice the (apparently) different roles played by the constant Ω in the definition of Böhm trees and in the trees obtained by cutting them. When we build a Böhm tree, Ω means that we get an unsolvable term. In the cutting process, Ω simply replaces any Böhm tree, i.e. any term. To better understand the meaning of Ω we can consider the cuttings of the Böhm trees as photographs of the λ-terms obtained by halting the computations after some finite number of steps. In this view Ω simply corresponds to a λ-term which is not in head normal form. This is exactly the role played by Ω in the definition of η-approximants of λ-terms (see Definition 9.3).

To formalize this idea, we can define the approximate Böhm tree of a λ-term M ($ABT(M)$) as follows:

- if M is in head normal form, i.e. $M \equiv \lambda x_1 \dots x_n.yN_1 \dots N_m$ $(n, m \geq 0)$, then:



- otherwise, i.e. if M is not in head normal form:

$$ABT(M) = \Omega.$$

We can extend the definition of Böhm trees to $\lambda\Omega$-terms by assuming:

$$BT(\lambda x_1 \dots x_n.\Omega M_1 \dots M_m) = \Omega,$$

where $n, m \geq 0$. Then an alternative and equivalent definition is

$$ABT(M) = BT(\alpha_\eta(M))$$

where α_η is the mapping associating an η-approximant to each λ-term (Definition 9.2).

For example, if $\mathbf{I} \equiv \lambda x.x$, it is easy to verify that the λ-terms $\mathbf{I}\Delta_3$, $\lambda x.x(\mathbf{I}x)(\mathbf{I}x)$, $\lambda x.xx(\mathbf{I}x)$ and $\lambda x.x(\mathbf{I}x)x$ are all reducible to Δ_3 and that their approximate Böhm trees are all the possible cuttings of $BT(\Delta_3)$. Notice that $ABT(\Delta_3) \equiv BT(\Delta_3)$, and clearly this holds for all terms in normal form.

In the case of $\mathbf{Y}$ we have:

$$\mathbf{Y} =_\beta \lambda f.f(\mathbf{Y}f) =_\beta \lambda f.f(f(\mathbf{Y}f)) =_\beta \lambda f.f(f(f(\mathbf{Y}f))) =_\beta \dots$$

and the approximate Böhm trees of these λ-terms are the cuttings of $BT(\mathbf{Y})$.

A tree obtained by cutting another tree carries at most the same information, so it is natural to introduce the following partial order on trees:

$$T \sqsubseteq T' \text{ iff } T \text{ results from } T' \text{ by cutting off some subtrees.}$$

The partial order on trees naturally induces a partial order on $\lambda\Omega$-terms as follows:

$$M \sqsubseteq N \text{ iff } BT(M) \sqsubseteq BT(N).$$

By definition we get that the approximate Böhm tree of a λ-term is less than or equal to the Böhm tree of that term, i.e.

$$ABT(M) \sqsubseteq BT(M) \text{ for all } \lambda\text{-terms } M.$$

Moreover the notion of Böhm tree is clearly invariant under β-convertibility of λ-terms, so we have:

$$\text{if } N =_\beta M \text{ then } ABT(N) \sqsubseteq BT(M).$$

On the other hand, it is easy to verify that, if the tree T is obtained by cutting $BT(M)$, then there is a λ-term N such that $N =_\beta M$ and $ABT(N) = T$. Since we immediately have that each Böhm tree is the least upper bound of its cuttings w.r.t. $\sqsubseteq$, we can conclude that

$$BT(M) = \bigsqcup\{ABT(N) \mid N =_\beta M\}.$$

Clearly there is a one-one correspondence between η-approximants and finite Böhm trees. Recalling the definition of the set $\mathcal{A}_\eta(M)$ of the η-approximants of M (Definition 9.3) we get

$$BT(M) = \bigsqcup\{BT(A) \mid A \in \mathcal{A}_\eta(M)\}.$$

We want to associate to each η-approximant a pair $\langle \textit{basis}; \textit{type}\rangle$ which carries the same information [32, 86]. We call this pair the ***principal pair*** of the η-approximant. We say that two pairs are compatible if they do not share common type variables.

Definition 13.2 *Let A be an η-approximant. The principal pair of A is inductively defined as follows.*

1. $pp(\Omega) = \langle \emptyset; \omega \rangle$.
2. *If* $pp(A) = \langle \Gamma, x{:}\,\sigma; \tau \rangle$, *then* $pp(\lambda x.A) = \langle \Gamma; \sigma \to \tau \rangle$.
3. *If* $pp(A) = \langle \Gamma; \tau \rangle$ *and x does not occur in Γ, then* $pp(\lambda x.A) = \langle \Gamma; \omega \to \tau \rangle$.
4. *If* $pp(A_i) = \langle \Gamma_i; \tau_i \rangle$ $(1 \leq i \leq m)$ *are compatible pairs*[10], *then* $pp(xA_1 \ldots A_m) = \langle \Gamma; \varphi \rangle$, *where* $\Gamma = \biguplus_{i=1}^{m} \Gamma_i \uplus \{x{:}\,\tau_1 \to \ldots \to \tau_m \to \varphi\}$ *and φ is fresh*[11].

For example we have
$pp(x) = \langle \{x : \varphi\}; \varphi \rangle$,
$pp(\Delta_3) = \langle \emptyset; \varphi \wedge \psi \wedge (\varphi \to \psi \to \chi) \to \chi \rangle$,
$pp(\Omega) = \langle \emptyset; \omega \rangle$,
$pp(\lambda f.f\Omega) = \langle \emptyset; (\omega \to \varphi) \to \varphi \rangle$,
$pp(\lambda f.f(f\Omega)) = \langle \emptyset; (\omega \to \varphi) \wedge (\varphi \to \psi) \to \psi \rangle$,
$pp(\lambda f.f(f(f\Omega))) = \langle \emptyset; (\omega \to \varphi) \wedge (\varphi \to \psi) \wedge (\psi \to \chi) \to \chi \rangle$.

Each principal pair is deducible for the corresponding η-approximant in the system BCD, i.e.

$$pp(A) = \langle \Gamma; \tau \rangle \textit{ implies } \Gamma \vdash_\eta A : \tau.$$

This can be easily verified by induction on the set of η-approximants.
If $A \equiv \Omega$ then $\vdash_\eta \Omega : \omega$ by axiom (ω).
If $A \equiv \lambda x.A'$ and $pp(A) = \langle \Gamma; \sigma \to \tau \rangle$ where $pp(A') = \langle \Gamma, x{:}\,\sigma; \tau \rangle$, then by induction $\Gamma, x{:}\,\sigma \vdash_\eta A' : \tau$, so we conclude $\Gamma \vdash_\eta A : \sigma \to \tau$ by rule $(\to \mathrm{I})$.
If $A \equiv \lambda x.A'$ and $pp(A) = \langle \Gamma; \omega \to \tau \rangle$ where $pp(A') = \langle \Gamma; \tau \rangle$, then by induction $\Gamma \vdash_\eta A' : \tau$. Since x does not occur free in Γ and A', then we may derive $\Gamma,\ x{:}\omega \vdash_\eta A' : \tau$ (as weakening is admissible): from this we get $\Gamma \vdash_\eta \lambda x.A' : \omega \to \tau$ by $(\to \mathrm{I})$.
If $A \equiv xA_1 \ldots A_m$ then $pp(A) = \langle \Gamma; \varphi \rangle$ where $\Gamma = \biguplus_{i=1}^{m} \Gamma_i \uplus \{x{:}\,\tau_1 \to \ldots \to \tau_m \to \varphi\}$ and $pp(A_i) = \langle \Gamma_i; \tau_i \rangle$ $(1 \leq i \leq m)$. Clearly $\Gamma \vdash_\eta x{:}\,\tau_1 \to \ldots \to \tau_m \to \varphi$ using axiom (var) and possibly rule $(\leq_\eta)$. By induction we get $\Gamma_i \vdash_\eta A_i : \tau_i$, which implies $\Gamma \vdash_\eta A_i : \tau_i$ for $(1 \leq i \leq m)$, since weakening is admissible in our system. Therefore we can deduce $\Gamma \vdash_\eta A : \varphi$ as follows:

$$\dfrac{\Gamma \vdash x{:}\,\tau_1 \to \cdots \to \tau_m \to \varphi \qquad \Gamma \vdash A_1{:}\,\tau_1}{\Gamma \vdash xA_1{:}\,\tau_2 \to \cdots \to \tau_m \to \varphi}\ (\to\mathrm{E})$$
$$\vdots$$
$$\dfrac{\Gamma \vdash xA_1 \ldots A_{m-1}{:}\,\tau_m \to \varphi \qquad\qquad \Gamma \vdash A_m{:}\,\tau_m}{\Gamma \vdash xA_1 \ldots A_m{:}\,\varphi}\ (\to\mathrm{E})$$

[10]This condition can always be satisfied by suitably choosing the names of the type variables.
[11]To be more precise, we associate a set of principal pairs to each η-approximant since principal pairs are unique only up to the names of type variables and different type variables are associated to different occurrences of term variables.

The name principal pair is justified by the fact (proved in [86]) that all types deducible for an η-approximant A can be obtained from $pp(A)$ using suitable operations.

We are now interested in studying when we can deduce the principal pair of an η-approximant for another η-approximant. We look first at some examples.

We have that $pp(\lambda f.f\Omega) = \langle \emptyset; (\omega \to \varphi) \to \varphi \rangle$, and we can deduce $\vdash_\eta \lambda f.f(f\Omega) : (\omega \to \varphi) \to \varphi$ as follows:

$$\dfrac{\dfrac{f:\omega \to \varphi \vdash f:\omega \to \varphi \quad (\text{var}) \qquad f:\omega \to \varphi \vdash f\Omega : \omega \quad (\omega)}{f:\omega \to \varphi \vdash f(f\Omega):\varphi}\,(\to E)}{\vdash \lambda f.f(f\Omega):(\omega \to \varphi) \to \varphi}\,(\to I)$$

Note that $\lambda f.f\Omega \sqsubseteq \lambda f.f(f\Omega)$. This example can be generalized, i.e. we can always deduce the principal pair of an η-approximant for a "better" (w.r.t. $\sqsubseteq$) η-approximant. This follows from the fact that ω is the only type deducible for Ω, and that ω can be deduced for any term. So we can state:

$$\text{if } A \sqsubseteq A' \text{ and } pp(A) = \langle \Gamma; \tau \rangle \text{ then } \Gamma \vdash_\eta A' : \tau. \qquad (12)$$

Let us consider now $pp(\lambda xy.xy) = \langle \emptyset; (\varphi \to \psi) \to \varphi \to \psi \rangle$. We have the following derivation:

$$\dfrac{x:\varphi \to \psi \vdash x:\varphi \to \psi \quad (\text{var})}{\vdash \lambda x.x:(\varphi \to \psi) \to \varphi \to \psi}\,(\to \text{I})$$

Also this example enlightens a general phenomenon: we can always deduce the principal pair of an η-approximant A for an η-approximant obtained by η-reducing A. This simply follows from the fact that BCD typings are preserved by η-reduction, as remarked in Section 8. We can then state:

$$\text{if } A \overset{*}{\to}_\eta A' \text{ and } pp(A) = \langle \Gamma; \tau \rangle \text{ then } \Gamma \vdash_\eta A' : \tau. \qquad (13)$$

Now if we want to put together (12) and (13) we need to consider a partial order on η-approximants which embodies both $\sqsubseteq$ and $\overset{*}{\to}_\eta$. So we define the relation $\sqsubseteq_\eta$ as the least partial order on η-approximants such that:

- $\Omega \sqsubseteq_\eta A$ for all $A \in \mathcal{A}$;
- $\lambda y.xA_1 \ldots A_m y \sqsubseteq_\eta xA_1 \ldots A_m$ whenever $y \notin FV(xA_1 \ldots A_m)$;
- if $A \sqsubseteq_\eta A'$, then $\lambda x.A \sqsubseteq_\eta \lambda x.A'$;
- if $A_i \sqsubseteq_\eta A'_i$ for $1 \le i \le m$, then $xA_1 \ldots A_m \sqsubseteq_\eta xA'_1 \ldots A'_m$.

Using $\sqsubseteq_\eta$ we can conclude

$$\text{if } A \sqsubseteq_\eta A' \text{ and } pp(A) = \langle \Gamma; \tau \rangle \text{ then } \Gamma \vdash_\eta A' : \tau. \qquad (14)$$

The interest of this last statement is that it can be reversed, i.e. we get:

Theorem 13.3 *If $pp(A) = \langle \Gamma; \tau \rangle$ and $\Gamma \vdash_\eta A'{:}\,\tau$ then $A \sqsubseteq_\eta A'$.*

Proof. We say that Γ is a *principal basis* if $\langle \Gamma; \tau \rangle = pp(A)$ for some A, τ. We show Theorem 13.3 by proving the more general statement:

If $pp(A) = \langle \Gamma; \tau \rangle$, $\Gamma \Subset \Gamma'$, Γ' is a principal basis, and $\Gamma' \vdash_\eta A'{:}\,\tau$ then $A \sqsubseteq_\eta A'$

by structural induction on A.
The case $A \equiv \Omega$ is trivial.
Let $A \equiv \lambda x.B$ and $pp(A) = \langle \Gamma; \sigma \to \tau \rangle$ where $pp(B) = \langle \Gamma, x{:}\,\sigma; \tau \rangle$. If $A' \equiv \lambda x.B'$ from $\Gamma' \vdash_\eta A'{:}\,\sigma \to \tau$, we get by the BCD Generation Lemma $\Gamma', x{:}\,\sigma \vdash_\eta B'{:}\,\tau$, then by induction $B \sqsubseteq_\eta B'$, so we conclude $A \sqsubseteq_\eta A'$. If A' is a λ-free term, then we consider the η-approximant $\lambda z.A'z$, where z is fresh. From $\Gamma' \vdash_\eta A'{:}\,\sigma \to \tau$ we get $\Gamma' \vdash_\eta \lambda z.A'z{:}\,\sigma \to \tau$ using rules $(\to \mathrm{E})$ and $(\to \mathrm{I})$. So we are in previous case and we get $A \sqsubseteq_\eta \lambda z.A'z$. We can conclude $A \sqsubseteq_\eta A'$ since $\lambda z.A'z \sqsubseteq_\eta A'$.
The case $A \equiv \lambda x.B$ and $pp(A) = \langle \Gamma; \omega \to \tau \rangle$ where $pp(B) = \langle \Gamma; \tau \rangle$ can be dealt similarly.
Let $A \equiv xA_1 \ldots A_m$ and $pp(A) = \langle \Gamma; \varphi \rangle$ where $\Gamma = \biguplus_{i=1}^m \Gamma_i \uplus \{x{:}\,\tau_1 \to \ldots \to \tau_m \to \varphi\}$, $pp(A_i) = \langle \Gamma_i; \tau_i \rangle$ $(1 \le i \le m)$, and φ is fresh. By the BCD Generation Lemma $\Gamma' \vdash_\eta A'{:}\,\varphi$ implies that A' is λ-free, i.e. $A' \equiv yA'_1 \ldots A'_{m'}$. We get (using again the BCD Generation Lemma) $\Gamma' \vdash_\eta y{:}\,\tau'_1 \to \ldots \to \tau'_{m'} \to \varphi$, and $\Gamma' \vdash_\eta A'_j{:}\,\tau'_j$ for $1 \le j \le m'$ for some $\tau'_1, \ldots, \tau'_{m'}$. Since Γ' is a principal basis, φ occurs only once in Γ' as rightmost subtype of an arrow type. Moreover the condition $\Gamma \Subset \Gamma'$ implies $x{:}\,\tau_1 \to \ldots \to \tau_m \to \varphi \in \Gamma'$. Therefore we get by the BCD Generation Lemma and Proposition 9.6 $x \equiv y$, $\tau_1 \to \ldots \to \tau_m \to \varphi \le \tau'_1 \to \ldots \to \tau'_{m'} \to \varphi$. The last relation in turn implies by Proposition 9.6 $m = m'$ and $\tau'_j \le \tau_j$ for $1 \le j \le m$. So we get $\Gamma' \vdash_\eta A'_j{:}\,\tau_j$, i.e. by induction $A_j \sqsubseteq_\eta A'_j$ for $1 \le j \le m$. We can conclude $A \sqsubseteq_\eta A'$. □

The interest of the previous Theorem is that it allows us to discriminate by typings λ-terms which have different Böhm trees. In order to show this, we need to relate the partial order $\sqsubseteq_\eta$ with the equality of Böhm trees. First notice that the isomorphism between the set of finite Böhm trees and the set of η-approximants makes $\sqsubseteq_\eta$ a partial order on finite Böhm trees. Now consider two arbitrary Böhm trees T and T': if for all the cuttings of T there is a "better" (w.r.t. $\sqsubseteq_\eta$) cutting of T' and vice versa, then T and T' coincide.

Let us take now two λ-terms M and N such that $BT(M) \neq BT(N)$. By the above we can find at least one η-approximant A such that $A \in \mathcal{A}_\eta(M)$ and $\forall B \in \mathcal{A}_\eta(N)$ $A \not\sqsubseteq_\eta B$. We claim that if $pp(A) = \langle \Gamma; \tau \rangle$, then $\Gamma \vdash_\eta M{:}\,\tau$ and $\Gamma \not\vdash_\eta N{:}\,\tau$. In fact $\Gamma \vdash_\eta A{:}\,\tau$ and $A \in \mathcal{A}_\eta(M)$ imply by the BCD Approximation Theorem $\Gamma \vdash_\eta M{:}\,\tau$. If we assume ad absurdum that $\Gamma \vdash_\eta N{:}\,\tau$ we get again by the BCD Approximation Theorem $\Gamma \vdash_\eta B{:}\,\tau$ for some $B \in \mathcal{A}_\eta(N)$. This implies $A \sqsubseteq_\eta B$ by Theorem 13.3, so we get a contradiction.

As last remark, we want to point out the meaning of this result when we consider the filter model $\mathcal{F}(\mathcal{T})$ discussed in Section 11. First notice that by the BCD Approximation Theorem and Theorem 11.4 two λ-terms that have the same Böhm tree, i.e. the same set of η-approximants, must be equal in this filter model. Now assume that two λ-terms have different Böhm trees: by the above there are Γ and τ such that $\Gamma \vdash_\eta M{:}\,\tau$ and $\Gamma \nvdash_\eta N{:}\,\tau$, so they are different in $\mathcal{F}(\mathcal{T})$. Therefore we can say that the λ-theory of this filter model coincides with the equality of Böhm trees, a result already proved in [84]. The feature of the present proof is that when two λ-terms have different Böhm trees we exhibit a compact element of the model which discriminates between them. In fact we get $\uparrow \tau \subseteq [\![M]\!]_{\xi_\Gamma}$ and $\uparrow \tau \not\subseteq [\![N]\!]_{\xi_\Gamma}$, where ξ_Γ is the term environment defined in the proof of Theorem 11.5.

14 Further Readings

The reader of the present paper has been assumed familiar with λ-calculus and Curry assignment system. Those who need background readings may start with [90, 54]. Barendregt book [15] is the fundamental monograph for the study of type-free λ-calculus, but it is hardly an introductory book. For λ-calculus with types an exhaustive survey is [16].

Intersection type assignment systems have been developed in a number of papers which have been cited in the previous sections. An extensive study of the syntactical properties of the several systems for intersection types is [8] (see also [9, 11]). In particular, the papers [32, 86, 85, 10] define principal type schemes for various intersection type disciplines. An exposition of the proof-theoretic properties of systems CDV and $\mathrm{CDV}_{\not\omega}$ used as tools for the syntactical study of type-free λ-calculus can be found in [61] (there called systems $\mathcal{D}\Omega$ and $\mathcal{D}$ respectively), which makes extensive use of the notion of saturated sets (adaptation of Tait computability predicate [89]).

The paper [17] gave rise to a number of studies relating intersection type theories to domain theory and λ-models. In particular, every inverse limit construction can be mimicked by a suitable EATS [29, 30, 34, 56, 75, 84]. Chapter 3 of [7] also treats intersection types and filter models in connection with Scott D_∞ λ-models. There also exist filter models that are isomorphic to Plotkin's and Engeler's models, respectively [76].

In his PhD thesis [3] Alessi gives an account of filter models as representations of domains. Moreover he studies certain pathologies of domains with respect to their representations, by defining a more basic category of models, which admit a good description in terms of intersection types. See also [4, 5]. In the same thesis, and in [57, 47], other categories of domains, like Girard's qualitative and quantitative domains, are studied using the intersection types.

All the above-mentioned models are models of the classical λ-calculus, but intersection-type disciplines are also suitable for describing models of the $\lambda\mathbf{I}$-calculus [9, 55], of the lazy λ-calculus [2, 18], of the call-by-value λ-calculus (introduced by Plotkin in [74]) [49, 78, 79], and of some extensions of the λ-calculus which include parallel features [21, 42, 43, 6].

The finitary descriptions of domains have been a major point of interest. Scott pioneered this topic with his Information Systems in [87]. Abramsky extended the theory to SFP domains in [1], framing it into the paradigm of Stone duality. A strikingly short, but good introduction to this subject is chapter 10 of [7]. We also suggest the reading of [20]: it is not exactly about filter models, but it gives a perspective to the webbed models (originated with Engeler models [48]), which are an alternative, though related, finitary description of λ-models. See also [75].

The use of trees to study the structure of λ-models dates to the 70's. Beside Böhm trees, introduced in Barendregt book [15], and recently considered in [41], Lévy-Longo trees have been known and studied at length: see [63, 64, 70]. A survey of various notions of trees and a study of their description via intersection type assignment is [14].

Since, given an arbitrary λ-term, it is undecidable whether it can be typed using intersection types, it is interesting to look for decidable restrictions, see [12, 36]. The inhabitation problem for intersection types (given a type decide whether there is a term typable with that type) is also undecidable [92], but a decidable subsystem has been devised in [62].

Extensions of intersection types with union and quantified types, both universal and existential, have been studied in [51, 66, 13, 23]. The completeness problem for such systems has been recently settled by Yokouchi in [94]. Intersection and union types turn out to be powerful enough to discriminate Lévy-Longo trees [45, 46].

Systems of intersection types à la Church are not trivial to define: they have been extensively studied in [81, 82, 83, 72, 73, 25, 26] and applied to type disciplines for object oriented languages in [27].

Lastly we mention that intersection and union types have been extensively used to prove properties of programs, like strictness analysis, detection of dead code, etc. Pioneers in this field were the PhD thesis of Benton [19] and Jensen [59] and the paper [35]. The recent PhD thesis by Mossin [69] and Damiani [40] contain the last developments and the references to the large literature on the subject.

References

[1] S. ABRAMSKY, "Domain Theory in Logical Form", *Annals of Pure and Applied Logics*, 51, 1991, 1-77.

[2] S. ABRAMSKY, C.-H.L. ONG, "Full Abstraction in the Lazy Lambda Calculus", *Information and Computation* 105, 1993, 159–267.

[3] F. ALESSI, *Strutture di Tipi, Teoria dei Domini e Modelli del λ-calcolo*, Ph.D. Thesis, Università di Torino, 1991.

[4] F. ALESSI, "Type Preorders", *LNCS* 787, Springer-Verlag, Berlin, 1994, 37-51.

[5] F. ALESSI, "The Category p-SFP", Technical Report UDMI/27/96/RR, Università di Udine, 1996.

[6] F. ALESSI, M.DEZANI-CIANCAGLINI, U. DE'LIGUORO, "A Convex Powerdomain over Lattices: its Logic and λ-Calculus", *Fundamenta Informaticae* 32, 1997, 193–250.

[7] R. AMADIO, P.L. CURIEN, *Domains and Lambda-Calculi*, Cambridge University Press, 1998.

[8] S. VAN BAKEL, *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems*, Ph.D. Thesis, Mathematisch Centrum, Amsterdam, 1993.

[9] S. VAN BAKEL, "Complete Restrictions of the Intersection Type Discipline", *Theoretical Computer Science* 102, 1992, 135–163.

[10] S. VAN BAKEL, "Principal Type Schemes for the Strict Type Assignment System", *J. of Logic and Computation* 3(6), 1993, 643–670.

[11] S. VAN BAKEL, "Intersection Type Assignment Systems", *Theoretical Computer Science* 151 (2), 1995, 385–436.

[12] S. VAN BAKEL, "Rank 2 Intersection Type Assignment in Term Rewriting Systems", *Fundamenta Informaticae*, 26(2):141-166, 1996.

[13] F. BARBANERA, M. DEZANI-CIANCAGLINI, U. DE'LIGUORO, "Intersection and Union Types: Syntax and Semantics", *Information and Computation* 119, 1995, 202–230.

[14] F. BARBANERA, M. DEZANI-CIANCAGLINI, F.-J. DE VRIES, "Types for Trees", *PROCOMET '98*, Chapman & Hall, London, 1998, 11-29.

[15] H.P. BARENDREGT, *The Lambda Calculus: Its Syntax and Semantics*, 2nd ed., North-Holland, Amsterdam, 1984.

[16] H.P. BARENDREGT, "The Lambda Calcull with Types", *Handbook of Logic in Computer Science* 2, Oxford Univ.Press, Oxford, 1992, 117-309.

[17] H.P. BARENDREGT, M. COPPO, M. DEZANI-CIANCAGLINI, "A Filter Lambda Model and the Completeness of Type Assignment", *J. Symbolic Logic* 48, 1983, 931–940.

[18] O.BASTONERO, A. PRAVATO, S. RONCHI DELLA ROCCA "Structures for Lazy Semantics", *PROCOMET '98*, Chapman & Hall, London, 1998, 30-48.

[19] P.N. BENTON, *Strictness Analysis of Lazy Functional Programs*, Ph.D. Thesis, University of Cambridge, Pembroke College, 1992.

[20] C. BERLINE, "From Computation to Foundations via Functions and Application: The λ-calculus and its Webbed Models", preprint, Équipe de Logique, Dépàrtement des Mathématiques, Université Denis Diderot (Paris VII), 1997 (to appear in *Theoretical Computer Science*).

[21] G. BOUDOL, "Lambda Calculi for (Strict) Parallel Functions", *Information and Computation*, 108, 1991, 51–127.

[22] F. CARDONE, M. COPPO, "Type Inference with Recursive Types. Syntax and Semantics", *Information and Computation*, 92(1), 1991, 48–80.

[23] F. CARDONE, M. DEZANI-CIANCAGLINI, U. DE'LIGUORO, "Combining Type Disciplines", *Annals of Pure and Applied Logic*, 66:197-230, 1994.

[24] A. Church, "A Formalization of the Simple Theory of Types", *J. of Symbolic Logic* 5, 1940, 56–68.

[25] A.B. Compagnoni, *Higher-Order Subtyping with Intersection Types*, Ph.D. Thesis, University of Nijmegen, 1995.

[26] A.B. Compagnoni, "Decidability of Higher-order Subtyping with Intersection Types", CSL'94, *LNCS* 933, Springer-Verlag, Berlin, 1995, 46–60.

[27] A.B. Compagnoni, B.C. Pierce, "Higher Order Intersection Types and Multiple Inheritance", *Mathematical Structures in Computer Science* 6, 1996, 469–501.

[28] M. Coppo, M. Dezani-Ciancaglini, "An Extension of the Basic Functionality Theory for the λ-Calculus ", *Notre Dame Journal of Formal Logic* 21(4), 1980, 685–693.

[29] M. Coppo, M. Dezani-Ciancaglini, F. Honsell, G. Longo, "Extended Type Structures and Filter λ-models", *Logic Colloquium '82*, North-Holland, Amsterdam, 1984, 241–262.

[30] M. Coppo, M. Dezani-Ciancaglini, G. Longo, "Applicative Information Systems", CAAP'83, *LNCS* 159, Springer-Verlag, Berlin, 1983, 35–64.

[31] M. Coppo, M. Dezani-Ciancaglini, P. Sallé, "Functional Characterization of Some Semantic Equalities inside λ-calculus", ICALP'79, *LNCS* 71, Springer-Verlag, Berlin, 1979, 133-146.

[32] M. Coppo, M. Dezani-Ciancaglini, B. Venneri, "Principal Type Schemes and λ-calculus Semantics", *To H.B.Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, New York, 1980, 535–560.

[33] M. Coppo, M. Dezani-Ciancaglini, B. Venneri, "Functional Characters of Solvable Terms", *Zeitschrift für Mathematische Logik*, 27, 1981, 45–58.

[34] M. Coppo, M. Dezani-Ciancaglini, M. Zacchi, "Type Theories, Normal Forms and D_∞-lambda-models", *Information and Computation* 72(2), 1987, 85–116.

[35] M. Coppo, A. Ferrari, "Type Inference, Abstract Interpretation and Strictness Analysis", *Theoretical Computer Science* 121, 1993, 113–144.

[36] M. Coppo, P. Giannini, "Principal Types and Unification For a Simple Intersection Type System" *Information and Computation* 122, 1995, 70–96.

[37] H.B. Curry, "Functionality in Combinatory Logic", *Proceedings of Natural Academy of Sciences U.S.A.* 20, 1934, 584–590.

[38] H.B. Curry, K. Feys, *Combinatory Logic*, North Holland, Amsterdam, 1958.

[39] D. van Dalen, *Logic and Structure*, Third Augmented Edition, Springer-Verlag, Berlin, 1994.

[40] F. Damiani, *Non-standard Type Inference for Functional Programs*, Ph.D. Thesis, Università di Torino, 1998.

[41] M. Dezani-Ciancaglini, B. Intrigila, M. Venturini-Zilli, "Böhm's Theorem for Böhm Trees", *ICTCS'98*, World Scientific Pub., Oxford, 1998, 1–23.

[42] M. Dezani-Ciancaglini, U. de'Liguoro, A.Piperno, "Filter Models for Conjunctive-Disjunctive Lambda-calculi", *Theoretical Computer Science* 170(1–2), 1996, 83–128.

[43] M. Dezani-Ciancaglini, U. de'Liguoro, A. Piperno, "A Filter Model for Concurrent Lambda-calculus", *Siam Journal on Computing* 27(5), 1998, 1376 - 1419.

[44] M. Dezani-Ciancaglini, I. Margaria "A Characterization of F-Complete Type Assignments", *Theoretical Computer Science* 45, 1986, 121–157.

[45] M. Dezani-Ciancaglini, J. Tiuryn, P. Urzyczyn, "Discrimination by Parallel Observers", *LICS'97*, IEEE Comp. Soc. Press, Los Alamitos, 1997, 396–407.

[46] M. Dezani-Ciancaglini, J. Tiuryn, P. Urzyczyn, "Discrimination by Parallel Observers: the Algorithm", *Information and Computation* 150(2), 1999, 153–186.

[47] P. Di Gianantonio, F. Honsell, "An Abstract Notion of Application", TLCA'93, *LNCS* 664, Springer-Verlag, Berlin, 1993, 124–138.

[48] E. Engeler, "Algebras and Combinators", *Algebra Universalis* 13(3), 1981, 289–371.

[49] L. Egidi, F. Honsell, S. Ronchi della Rocca, "Operational, Denotational and Logical Descriptions: A Case Study", *Fundamenta Informaticae* 16(2), 1992, 149–170.

[50] G.K. Gierz, K.H. Hoffmann, K. Keimel, J.D. Mislove, D.S. Scott, *A Compendium of Continuous Lattices*, Springer-Verlag, Berlin, 1980.

[51] S. Hayashi, "Singleton, Union, and Intersection Types for Program Extraction", *Information and Computation* 109, 1994, 174–210.

[52] J.R. Hindley, "Coppo-Dezani Types do not Correspond to Propositional Logic", *Theoretical Computer Science* 28, 1984, 235–236.

[53] J.R. Hindley, "Types with Intersection, an Introduction", *Formal Aspects of Computing* 4, 1992, 470–486.

[54] J.R. Hindley, J.P. Seldin, *Introduction to Combinators and Lambda-calculus*, Cambridge Univ. Press, Cambridge, 1986.

[55] F. Honsell, M. Lenisa, "Some Results on the Full Abstraction Problem for Restricted Lambda Calculi", MFCS'93, *LNCS* 711, Springer-Verlag, Berlin, 1993, 84-104.

[56] F. Honsell, S. Ronchi Della Rocca, "An Approximation Theorem for Topological Lambda Models and the Topological Incompleteness of Lambda Calculus", *J. of Computer System Science* 45, 1992, 49–75.

[57] F. Honsell , S. Ronchi Della Rocca, "Reasoning about Interpretations in Qualitative Lambda Models", *PROCOMET'90*, North-Holland, Amsterdam, 1990, 505–521.

[58] W. Howard, "The Formulae-as-types Notion of Construction", *To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, New York, 1980, 479–490.

[59] T. P. Jensen, *Abstract Interpretation in Logical Form*, Ph.D. Thesis, University of London, Imperial College, 1992.

[60] J.W. Klop, "Combinatory Reduction Systems", Math. Center Tracts 129, Amsterdam, 1980.

[61] J.L. Krivine, *Lambda-calcul, Types et Modèles*, Masson, Paris, 1990.

[62] T. Kurata, M. Takahashi, "Decidable Properties of Intersection Type Systems", TLCA'95, *LNCS* 902, 1995, 297–311.

[63] J.J. Lévy, "An Algebraic Interpretation of the λ-β-**K**-calculus and an Application of the Labelled λ-calculus", *Theoretical Computer Science* 2(1), 1976, 697–114.

[64] G. Longo, "Set Theoretical Models of Lambda Calculus: Theory, Expansions and Isomorphisms", *Annals of Pure and Applied Logics* 24, 1983,153–188.

[65] E.G.K. Lopez-Escobar, "Proof Functional Connectives", *LNM* 1130, Springer-Verlag, Berlin, 1985, 352–366.

[66] I. Margaria, M Zacchi, "Principal Typing in a $\forall\wedge$-discipline", *Journal of Logic and Computation* 5(3), 1995, 367–381.

[67] R. Milner, M. Tofte, *Commentary on Standard ML*, MIT Press, 1990.

[68] R. Milner, M. Tofte, R. Harper, *The Definition of Standard ML*, MIT Press, 1991.

[69] C. Mossin, *Flow Analysis of Typed Higher-Order Programs*, Ph.D. Thesis, DIKU, University of Copenhagen, 1997.

[70] C.-H.L. Ong, *The Lazy λ-calculus: an Investigation into the Foundations of Functional Programming*, Ph.D. Thesis, University of London, 1988. (Also Prize Fellowship Dissertation, Trinity College, Cambridge.)

[71] L.C. Paulson, *ML for the Working Programmer*, Cambridge University Press, 1991.

[72] B.C. Pierce, *Programming with Intersection Types and Bounded Polymorphism*, CMU-CS-91-205, Ph.D. Thesis, Carnegie Mellon University, 1991.

[73] B.C. Pierce, "Intersection Types and Bounded Polymorphism", TLCA'93, *LNCS* 664, 1993, 346–360.

[74] G.D. Plotkin, "Call-by-name, Call-by-value and the λ-calculus", *Theoretical Computer Science* 1, 1975, 125–159.

[75] G.D. PLOTKIN, "Set-Theoretical and other Elementary Models of the λ-calculus",*Theoretical Computer Science* 121, 1993, 351–410.

[76] G.D. PLOTKIN, "A Semantics for Static Type Inference", *Information and Computation* 109, 1994, 256–299.

[77] G. POTTINGER "A Type Assignment for Strongly Normalizable λ-terms", *To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, New York, 1980, 561-577.

[78] A. PRAVATO, *Categorical Models of Untyped Lambda Calculi: a Monoïdal Approach*, Ph.D. Thesis, Università di Torino, 1997.

[79] A. PRAVATO, S. RONCHI DELLA ROCCA, L. ROVERSI, "The Call-by-value Lambda-calculus: a Semantic Investigation", *Mathematical Structures in Computer Science* 9(5), 1999, 617- 650.

[80] D. PRAWITZ, *Natural Deduction – A Proof-Theoretical Study*, Almqvist & Wiksell, Stockholm, 1965.

[81] J.C. REYNOLDS, "Preliminary Design of the Programming Language Forsythe", Repost CMU-CS-88-159, Carnegie Mellon University, 1988.

[82] J.C. REYNOLDS, "Syntactic Control of Interference Part 2", ICALP'89, *LNCS* 372, Springer-Verlag, Berlin 1989, 704–722.

[83] J.C. REYNOLDS, "The Coherence of Language with Intersection Types", TACS'91, *LNCS* 526, Springer-Verlag, Berlin 1991, 675–700.

[84] S. RONCHI DELLA ROCCA, "Basic Lambda Calculus", Notes of the *Advanced School on Typed Lambda Calculus and Functional Programming*, Università di Udine, 1994.

[85] S. RONCHI DELLA ROCCA, "Principal Type Scheme and Unification for Intersection Type Discipline", *Theoretical Computer Science* 59, 1988, 1–29.

[86] S. RONCHI DELLA ROCCA, B. VENNERI, "Principal Type Schemes for an Extended Type Theory", *Theoretical Computer Science* 28, 1984, 151-169.

[87] D. SCOTT, "Domains for Denotational Semantics", ICALP'82, *LNCS* 140, Springer-Verlag, Berlin 1982, 577–613.

[88] V. STOLTENBERG-HANSEN, I. LINDSTRÖM, E.R. GRIFFOR, *Mathematical Theory of Domains*, Cambridge University Press, 1994.

[89] W.W. TAIT, "Intensional Interpretation of Functionals of Finite Type", *J. of Symbolic Logic* 32, 1967, 198-212.

[90] M. TAKAHASHI, "A Primer on Proofs and Types", *this volume*, 1–44.

[91] A.S. TROELSTRA, ed., *Metamathematical Investigations of Intuitionistic Arithmetic and Analysis, LNM* 344, Springer-Verlag, Berlin, 1973.

[92] P. URZYCZYN, "The Emptiness Problem for Intersection Types", *LICS'94*, IEEE Comp. Soc. Press, Los Alamitos, 1994, 300–309.

[93] B. VENNERI, “Intersection Types as Logical Formulae”, *J. of Logic and Computing* 4(2), 1994, 109–124.

[94] H. YOKOUCHI, “Completeness of Type Assignment Systems with Intersection, Union, and Type Quantifiers”, *LICS'98*, IEEE Comp. Soc. Press, Los Alamitos, 1998, 368–379.